Filip De Turck Wolfgang Kellerer
George Kormentzas (Eds.)

Managing Large-Scale Service Deployment

19th IFIP/IEEE International Workshop on Distributed Systems:
Operations and Management, DSOM 2008
Samos Island, Greece, September 22-26, 2008
Proceedings

Springer

Volume Editors

Filip De Turck
Ghent University
Dept. of Information Technology - IBBT - IMEC
Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium
E-mail: filip.deturck@intec.ugent.be

Wolfgang Kellerer
Ubiquitous Networking Research
DoCoMo Communications Laboratories Europe GmbH
Landsberger Str. 312, 80687 Munich, Germany
E-mail: kellerer@docomolab-euro.com

George Kormentzas
Universiy of the Aegean
Department of Information and Communication Systems Engineering
83200 Karlovassi, Greece
E-mail: gkorm@aegean.gr

Library of Congress Control Number: 2008934860

CR Subject Classification (1998): C.2, H.3.4, E.1, H.2.4, D.4.4

LNCS Sublibrary: SL 5 – Computer Communication Networks
and Telecommunications

ISSN 0302-9743
ISBN-10 3-540-85999-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-85999-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12522026 06/3180 5 4 3 2 1 0

Preface

This volume of the *Lecture Notes in Computer Science* series contains all papers accepted for presentation at the *19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2008)*, which was held September 25-26, 2008 on the island of Samos, Greece. DSOM 2008 was the 19th event in a series of annual workshops. It followed in the footsteps of previous successful meetings, the most recent of which were held in San José, California, USA (DSOM 2007), Dublin, Ireland (DSOM 2006), Barcelona, Spain (DSOM 2005), Davis, California, USA (DSOM 2004), Heidelberg, Germany (DSOM 2003), and Montreal, Canada (DSOM 2002). The goal of the DSOM workshops is to bring together researchers from industry and academia working in the areas of networks, systems, and service management, to discuss recent advances and foster future growth. In contrast to the larger management conferences, such as IM (Integrated Network Management) and NOMS (Network Operations and Management Symposium), DSOM workshops have a single-track program in order to stimulate more intense interaction among participants.

The theme of DSOM 2008 was "Managing Large-Scale Service Deployment" focusing both on management of overlay networks and on virtualized service infrastructures. The concepts of abstract overlays and virtualization constitute key contributors for efficient large-scale service deployment and testing. Scalable abstract overlay networks accompanied with appropriate management techniques offer flexibility for future service deployment and consumption with high quality of experience. Virtualization decouples service substantiation from its actual realization over networking and IT infrastructures enabling wide-scale deployment of services. At the same time, DSOM 2008 continued its tradition of giving a platform to papers that address general topics related to the management of distributed systems. As a result, DSOM 2008 included sessions on decentralized and peer-to-peer management, operations and tools, security and trust, and measurements, monitoring and diagnosis.

Like the previous three DSOM workshops, DSOM 2008 was co-located with several related events as part of the 4th International Week on Management of Networks and Services (Manweek 2008). The other events were the 11th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS 2008), the 8th IEEE International Workshop on IP Operations and Management (IPOM 2008), the Third IEEE International Workshop on Modeling Autonomic Communications Environments (MACE 2008), the 4th IEEE/IFIP International Workshop on End-to-End Virtualization and Grid Management (EVGM 2008), and the 5th International Workshop on Next-Generation Networking Middleware (NGMN 2008). Co-locating these events provided the opportunity for an exchange of ideas between research communities that work on related topics, allowing participants to forge links and exploit synergies.

DSOM 2008 attracted a total of 38 paper submissions by authors from 20 different countries. Each paper received at least three reviews by experts in the field. A total of 14 submissions were finally accepted into the program as full papers.

DSOM 2008 owes its success in large part to a dedicated community of researchers from academia and industry that has formed over many years. First and foremost, we want to thank the authors of the submitted papers–without them, there would be no program. We also want to thank the members of the Technical Program Committee and the additional reviewers for their constructive and detailed reviews. A big "thank you" goes to Tom Pfeifer, our Publications Chair, who played a big part in creating these proceedings. Finally, we want to thank our patrons, HP, FP6 IST-UNITE project and FP7 ICT-HURRICANE project, whose financial support was essential to making DSOM 2008 a great event.

September 2008

Filip De Turck
Wolfgang Kellerer
George Kormentzas

DSOM 2008 Organization

Manweek 2008 Chair

George Kormentzas	University of the Aegean, Greece

Manweek 2008 Vice Chair

Francisco Guirao	European Commission

Manweek 2008 Advisors

Raouf Boutaba	University of Waterloo, Canada
Brendan Jennings	Waterford Institute of Technology, Ireland
Sven van der Meer	Waterford Institute of Technology, Ireland

DSOM 2008 Technical Program Committee

Issam Aib	University of Waterloo
Javier Baliosian	University of the Republic of Uruguay
Claudio Bartolini	HP Laboratories
Raouf Boutaba	University of Waterloo
Nevil Brownlee	The University of Auckland
Marcus Brunner	NEC Europe Ltd.
Omar Cherkaoui	University of Quebec in Montreal
Alexander Clemm	Cisco Systems
Thierry Coupaye	France Telecom
Luca Deri	ntop.org
Yixin Diao	IBM Research
Philippe Dobbelaere	Alcatel-Lucent
Gabi Dreo Rodosek	University of Federal Armed Forces
Metin Feridun	IBM Research
Olivier Festor	INRIA Nancy - Grand Est
Stéphane Frénot	INRIA Amazones
Alex Galis	University College London
Yacine Ghamri-Doudane	LRSM - ENSIIE
Kurt Geihs	Universität Kassel
Erol Gelenbe	Imperial College London
Masum Hasan	Cisco Systems
Heinz-Gerd Hegering	Leibniz Supercomputing Center
James Hong	POSTECH
Cynthia Hood	Illinois Institute of Technology
Yangcheng Huang	Ericsson
Gabriel Jakobson	Altusys Corp.
Brendan Jennings	TSSG, Waterford Institute of Technology
Alexander Keller	IBM Global Technology Services

Yoshiaki Kiriha	NEC
Frédéric Le Mouël	INRIA Amazones / INSA Lyon
David Lewis	Trinity College Dublin
Hong Li	Intel Corporation
Jorge Lopez de Vergara	Universidad Autonoma de Madrid
Emil Lupu	Imperial College
Hanan Lutfiyya	University of Western Ontario
Antonio Manzalini	Telecom Italia
Saverio Niccolini	NEC Europe Ltd.
Jose-Marcos Nogueira	UFMG
Declan O'Sullivan	Trinity College Dublin
Evaggelos Pallis	Centre for Technological Research of Crete
Symeon Papavassiliou	National Technical University of Athens
Luciano Paschoal Gaspary	UFRGS
Fotini-Niovi Pavlidou	Aristotle University of Thessaloniki
Aiko Pras	University of Twente
Thierry Pollet	Alcatel-Lucent
Juergen Quittek	NEC Europe Ltd.
Danny Raz	Technion
Akhil Sahai	VMWare Inc, USA
Jürgen Schönwälder	Jacobs University Bremen
Joan Serrat	Universitat Politécnica de Catalunya
Adarsh Sethi	University of Delaware
Rolf Stadler	KTH Stockholm
Radu State	LORIA - INRIA Lorraine
Burkhard Stiller	University of Zurich and ETH Zurich
Sven van der Meer	Waterford Institute of Technology
John Vicente	Intel Corporation
Vincent Wade	Trinity College Dublin
Tim Wauters	University of Ghent
Carlos Westphall	Federal University of Santa Catarina
S. Felix Wu	University of California at Davis
Geoffrey Xie	Naval Postgraduate School, Monterey
George Xylomenos	Athens University of Economics and Bussiness
Yang Richard Yang	Yale University
Makoto Yoshida	The University of Tokyo
Lisandro Z. Granville	UFRGS
Xiaoyun Zhu	Hewlett Packard Labs

Additional Reviewers

Ioannis Anagnostopoulos	University of the Aegean
Tiago Fioreze	University of Twente
Fermín Galán Márquez	Telefónica I+D
Alberto Gonzalez	KTH Royal Institute of Technology
Eduardo Grampin	Universidad de la Republica de Uruguay

Dan Jurca	KTH Royal Institute of Technology
Eleftherios Koutsoloukas	National Technical University of Athens
Anh Le	University of Waterloo
Angelos Lenis	National Technical University of Athens
Christos Politis	Kingston University
Vassiliki Pouli	National Technical University of Athens
Anna Sperotto	University of Twente
Yong Xiang	University of Waterloo

Table of Contents

SOON: A Scalable Self-organized Overlay Network for Distributed Information Retrieval

Juan Li and Son Vuong

Computer Science Department, University of British Columbia
2366 Main Mall, Vancouver, B.C., Canada
{juanli, vuong}@cs.ubc.ca

Abstract. Locating desirable resources and information from a large-scale distributed system such as P2P system and grid is a very important issue. However, the distributed, heterogeneous, and unstructured nature of the system makes this issue very challenging. In this paper, we propose Self-Organized Overlay Network (SOON), an unstructured P2P overlay architecture, to facilitate sharing and searching semantically heterogeneous contents. In particular, we have proposed a semantics-aware topology construction method to group nodes sharing similar semantics together to create small-worlds. For this purpose, we have designed an algorithm to extract a node's ontology summary and use that summary to compute the semantic similarity between nodes. With this semantic similarity defined, nodes are grouped accordingly, forming semantic virtual domains and clusters. Resource information integration and searching can be efficiently performed on top of this topology.

Keywords: Overlay network, P2P system, Semantic Web, topology.

1 Introduction

A widely-held belief pertaining to social networks is that any two people in the world are connected via a chain of six acquaintances (*six-degrees of separation*) [1]. The quantitative study of the phenomenon started with Milgram's experiments [2]. Milgram's experiments illustrated that individuals with only local knowledge of the network (i.e., their immediate acquaintances) may successfully construct acquaintance chains of short length, leading to networks with "small-world" characteristics. In such a network, a query can be forwarded along acquaintance chains taking it closer to the destination. Randomized network constructions that model the small-world phenomenon have recently received considerable attention. To model the routing aspects of the small-world phenomenon, Kleinberg constructed a family of random graphs [3]. He considered a 2D $n \times n$ grid with n^2 nodes. Each node is equipped with a small set of "local" contacts and one "remote" contact drawn from a harmonic distribution. With greedy routing, the path-length between any pair of nodes is $O(log^2n)$ hops, with high probability.

Small-world networks exhibit special properties, namely, a small average diameter and a high degree of clustering. A small diameter corresponds to a small separation between peers, while high clustering signals tight communities. Small-world graphs contain inherent community structure, where similar nodes are grouped together in some meaningful way. Intuitively, a network satisfying the small-world properties would allow peers to reach each other via short paths while maximizing the efficiency of communication within the clustered communities.

We draw inspiration from small-world networks and organize nodes in our system to form a small-world topology, particularly from a semantic perspective. Our objective is to make the system's dynamic topology match the semantic clustering of peers, i.e., there is a high degree of semantic similarity between peers within the clustered community; this would allow queries to quickly propagate among relevant peers as soon as one of them is reached.

In our overlay network, SOON, peers use their ontology summary to represent their expertise. Unlike most existing systems, SOON does not assume a global ontology but heterogeneous ontologies. We have designed a novel algorithm to compute the semantic similarity between two nodes in the network; then we use the semantic similarity as the metric to organize the network topology. Nodes are loosely structured in this network. Each of them keeps track of a set of neighbors and organizes these neighbors into a multi-resolution neighborhood according to their semantic similarities. This way, the overlay network topology is reconfigured with respect to peers' semantic properties, and peers with similar ontologies are close to each other. Information can be integrated and discovered through nodes' current neighbors, rather than by contacting some central hubs or virtual central hubs, such as Distributed Hash Tables (DHTs). This architecture combines the efficiency and scalability of structured overlay network with the connection flexibility of unstructured networks. It achieves full distribution, high scalability, and robustness.

2 Semantic Metadata

Metadata, the data about data, is a crucial element of a sharing and discovering infrastructure. An effective way of locating information of interest within large-scale information intensive environments is providing and managing metadata about the information. More important, metadata should be able to express the meaning of the information. An ontology, "a shared and common understanding of a domain that can be communicated between people and application systems", as considered in modern knowledge engineering [4], is precisely intended to convey that kind of shared understanding. An ontological representation defines concepts and relationships. It sets the vocabulary, properties, and relationships for concepts. The elements accumulate more meaning by the relationships they hold and the potential inferences that can be made by those relationships. This capability of formal ontologies to convey relationships and axioms make them ideal vehicles for describing the vocabulary for metadata statements, providing a rich formal semantic structure for their interpretation. Therefore, we use ontologies to represent information metadata semantics. To cope with the openness and extensibility requirements, we adopt two W3C recommendations, the Resource Description Framework (RDF) and the Web Ontology Language (OWL), as our ontology languages.

In our system the ontology knowledge is represented by OWL-DL and is separated into two parts: the terminological box (T-Box) and the assertion box (A-Box) as defined in the description logic terminology. The T-Box is a finite set of terminological axioms, which includes all axioms for concept definition and descriptions of domain structure, for example a set of classes and properties. The A-Box is a finite set of assertional axioms, which includes a set of axioms for the descriptions of concrete data and relations, for example, the instances of the classes defined in the T-Box. Generally speaking, there are many more A-Box instances than T-Box concepts. Separating the T-Box and A-Box enables different coarse-grained knowledge indexing, thus increasing the scalability of the system.

3 Semantic Similarity

To organize peers according to their semantic properties, we need a metric to measure peers' ontology similarity. There has been extensive research [6, 7, 8] focusing on measuring the semantic similarity between two objects in the field of information retrieval and information integration. However their methods are very comprehensive and computationally expensive. In this paper, we propose a simple method to compute the semantic similarity between two peers; this can easily be replaced with other advanced functions for a complex system.

3.1 Ontology Signature Set (OSS)

To measure the semantic similarity between peers, we need to extract each peer's semantic characteristics. The T-Box part of an ontology defines high-level concepts and their relationships like the schema of a database. It is a good abstraction of the ontology's semantics and structure. Therefore, we use keywords of a nodes' T-Box ontology as its ontology summary. For each node, we extract the class and property labels from its T-Box ontology, and put them into a set. This set is called this node's Ontology Signature Set (OSS). We can measure the similarity of two ontologies by comparing the elements of their OSSs. However, a semantic meaning may be represented by different labels in different ontologies, while it is also possible that the same literal label in different ontologies means totally different things. Ontology comparison based on primitive OSSs may not yield satisfying results. One improvement is to extend each concept with its semantic meanings, so that semantically related concepts would have overlaps. Based on this intuition, we use the lexical database, WorldNet [5], to extend the OSS to include words which are semantically related to the concepts from the original set.

WordNet is conceived as a machine-readable dictionary. It structures lexical information in terms of word meanings. WordNet maps word forms in word senses using the syntactic category as a parameter. Words of the same syntactic category that can be used to express the same meaning are grouped into a single synonym set, called *synset*. For example, the noun "computer" has a synset: {computer, data processor, electronic computer, information processing system}. An intuitive idea of extending an OSS is to extend each concept with its synset, i.e., its synonyms. In this way, two semantically related ontologies would have common WordNet terms in their extended OSSs. Besides synonyms, WordNet also includes other lexical semantic relations, such as *is-a*, *kind-of*, *part-of*. Among these relations, *is-a* (represented by hyponym/hypernym in WordNet) is the most important relationship; it explains a concept by a more general concept. Therefore, we also extend OSS concepts with their hypernyms.

After extension, an OSS may get a large number of synonyms for each concept. However, not all of these synonyms should be included in the set, because each concept may have many senses (meanings), and not all of them are related to the ontology context. Having unrelated senses in the OSS will diminish the accuracy of measuring the semantic similarity; thus we have to prune the expanded OSS to exclude those unrelated terms. A problem causing the ambiguity of concepts in OSS is that the extension does not make use of any relations in the ontology. Relations between concepts are important clues to infer the semantic meanings of concepts, and they should be considered when creating the OSS. Therefore, we utilize relations between the concepts in an ontology to further refine the semantic meaning of a particular concept. Only words with the most appropriate senses are added to the OSS. Since the dominant semantic relation in an ontology is the *subsumption* relation, we use the *subsumption* relation and the sense disambiguation information provided by WordNet to refine OSSs. It is based on a principle that a concept's semantic meaning should be consistent with its super-class's meaning. We use this principle to remove those inconsistent meanings. The refined algorithm to extend the primitive OSS is illustrated with the pseudocode in Fig. 1.

```
/* This algorithm generates the refined ontology signature
set OSS for an ontology, O */

createOss(Ontology O)
{
   OSS={};
   for each c ∈ {concepts of ontology O} do
        pc is parent concept of c
        add c, pc to oss
        for each Sc ∈ {senses of c} do
          Hc={hypernyms of Sc}
          for each Spc ∈ {senses of pc) do
              if Hc ∩ Spc !=null
                 add Sc,Spc to OSS
}
```

Fig. 1. A refined algorithm to create the Ontology Signature Set of an ontology O

The algorithm in Fig.1 creates the refined OSS of an ontology by adding the appropriate sense set of each ontology concept based on the *sub-class/super-class* relationships between the parent concepts and child concepts. For every concept in an ontology, we check each of its senses; if a sense's hypernym has overlap with this concept's parent's senses, then we add this sense and the overlapped parent's sense to the OSS set. In this way we can refine the OSS and reduce imprecision. Possible improvements could be obtained by using other relations in the ontology.

3.2 Peer Semantic Similarity

To compare two ontologies, we define an ontology similarity function based on the refined Ontology Signature Set. The definition is based on Tversky's "Ratio Model" [9] which is evaluated by set operations and is in agreement with an information-theoretic definition of similarity [10].

Definition 1: Assume A and B are two peers, and their extended Ontology Signature Sets are S(A) and S(B) respectively. The semantic similarity between peer A and peer B is defined as:

$$sim(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cap S(B)| + \alpha |S(A) - S(B)| + \beta |S(B) - S(A)|}$$

In the above equations, "$\cap$" denotes set intersection, "–" is set difference, while "||" represents set cardinality, "α" and "β" are parameters that provide for differences in focus on the different components. The similarity *sim*, between *A* and *B*, is defined in terms of the semantic concepts common to OSS of *A* and *B*: $S(A)\cap S(B)$, the concepts that are distinctive to *A*: $S(A)–S(B)$, and the features that are distinctive to *B*: $S(B) – S(A)$. With the similarity measure specified, we have the following definition:

Definition 2: Two nodes, node A and node B are said to be semantically equivalent if their semantic similarity measure, sim(A,B) equals to 1 (implying sime(B,A)=1 as well). Node A is said to be semantically related to node B, if sim(A,B) exceeds the user-defined similarity threshold t ($0<t\leq 1$). Node A is semantically unrelated to node B if sim(A,B)<t.

4 Self-organized Semantic Small-World Overlay

We follow the idea of the Kleinberg experiment to construct the semantic small-world network. In Kleinberg's experiment each node keeps many short-range contacts, as well as a small number of long-range contacts. In our system, a node distinguishes three kinds of neighbors based on their semantic similarity. A peer A's neighbor, B, can be one of these three types: (1) zero-distance neighbor (or semantically equivalent neighbor), if $sim(A,B)=1$, (2) short-distance neighbor (or semantically related neighbor) if $sim(A,B)\geq t$ ($0<t<1$ is A's semantic threshold), (3) long-distance neighbor (or semantically unrelated neighbor) if $sim(A,B)<t$. A node always tries to find as many close neighbors as possible, but it also keeps some long distance neighbors to reach out to other ontological clusters.

Nodes in the system randomly connect to each other through these three types of neighbors. They produce a semantically clustered small-world topology as shown in Fig.2. The clustered structure is not flat but multi-layered; nodes with similar ontological topics, i.e., short-distance neighbors, form a domain (a region formed by nodes with the same shape in the figure); inside the domain, nodes may create smaller clusters (sub-regions in a domain with same color) if they share the same ontology schema. For example, all peers in the medical domain are interested in medically related information. They may be interested in different aspects of the medical resources, and they may use different ontologies to describe their resources. They connect with each other through short-distance links. Inside the medical domain, nodes further organize themselves to finer-grained clusters based on their ontologies. For example, node N_1, N_2, N_5, N_8 and N_{11} use the same ontology O_3 (e.g., a medical ontology, SNOMED-RT [26]). Clusters and domains do not have fixed boundaries; they are formed by randomly connecting relevant nodes.

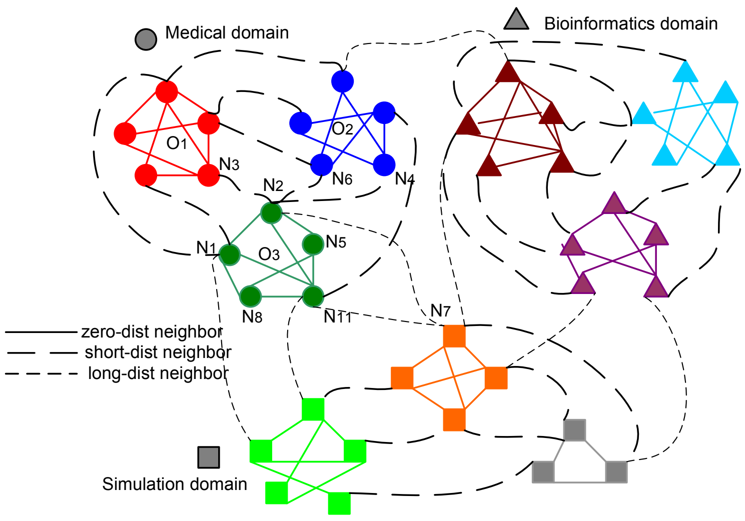


Fig. 2. The semantic small-world network topology

The construction of an ontology-based topology is a process of finding semantically related neighbors. A node joins the network by connecting to one or more bootstrapping neighbors. Then the joining node issues a neighbor-discovery query, and forwards the query to the network through its bootstrapping neighbors. When a node *N* receives a neighbor-discovery query *Q* which tries to find neighbors for a new joining node *X*, *N* computes the semantic similarity between *X* and itself. If *N* is semantically related to *X*, *N* will send a reply to *X*. If the query's TTL does not expire, *N* computes the semantic similarity between *X* and each of its neighbors, and forwards the query to semantically related neighbors. If no semantically related neighbors are found, the query will be forwarded to *N*'s long-distance neighbors.

With the semantic small-world topology constructed, information discovery can be efficiently performed. In most cases, a discovery query can be answered within the querying node's local domain, because queries reflect the querying node's ontology interest, and semantically related nodes are within the neighborhood of the querying node. When a node issues (or receives) a query, it first chooses its zero-distance neighbors to forward the query inside the local cluster. Since they use the same ontology, the zero-distance neighbors are the best candidates to forward the query to. Another important step in query processing is reformulating a peer's query over other peers on the available semantic paths. Starting from the querying peer, the query is reformulated based on the inter-ontology mapping over the querying peer's short-distance neighbors, then over their short-distance neighbors, and so on until the query TTL expires. Interested readers can refer our previous work [27] for the inter-ontology mapping schemes. Sometimes, users may want to locate resources in other semantic domains. In this case, they would first locate the related domain using the long distance-neighbors.

6 Experiments

We have performed extensive simulation experiments to evaluate the performance of our overlay network structure.

6.1 Setup

The test data is artificially generated. The T-Box ontologies are generated first, and then individuals are created by instantiating classes. We assume for simulation purposes that ontologies and queries are associated with a specific domain, and all ontologies in the same domain have ontology mappings defined in advance. The simulation is initialized by injecting nodes one by one into the network until a certain network size has been reached. After the initial topology is created, a mixture of joins, leaves, and queries are injected into the network based on certain ratios. The proportion of join to leave operations is kept the same to maintain the network at approximately the same size. Inserted nodes start functioning without any prior knowledge.

For comparisons, we simulate our SOON overlay in conjunction with the learning-based ShortCut overlay [11] and a random-walk based simple Gnutella overlay [25]. The ShortCut overlay, as will be described in the related work, is chosen as one comparison reference since it is simple yet effective, and many popular applications (e.g., [11], [12], [13], [14]) use this overlay as their basic routing overlay. Moreover, it is comparable to our network in the sense that it creates clusters on top of the unstructured network. Flooding-based Gnutella was chosen as another reference

Table 1. Parameters used in the simulations

Parameter	**Range and default value**
network size	2^9~2^{15} default: 10,000
initial neighbors (node degree)	5
average node degree	14
TTL	1~20 default 9
resource-discovery query walkers	3 (propagate exponentially)
neighbor-discovery query walkers	2 (propagate linearly)
ontology domains	1~10 default: 8
ontology schemas per domain	1~10 default:8
resources per node	1~10
die/leave probability per time slice per node	0-21%, 3% default
query probability per time slice per node	5%
sample of nodes to compute diameter	5%

network for its simplicity and prevalence, which, in fact, made it a widely used baseline for many previous research efforts. The simulation parameters and their default values are listed in Table 1.

6.2 Results and Discussion

Emergence of the small-world

As discussed, the topology of the peer network is a crucial factor determining the efficiency of the search system. We expect that the SOON semantic neighbor discovery scheme will transform the topology into a small-world network. To verify this transformation, we examine two network statistics, the *clustering coefficient* (*CC*) and the *average network path length (APL)*, as indicators of how closely the topology has approached a "small-world" topology. The *CC* is a measure of how well connected a node's neighbors are with each other. The *CC* of a node is the ratio of the number of existing edges and the maximum number of possible edges connecting its neighbors. The *APL* is defined as the average shortest path across all pairs of nodes. The *APL* corresponds to the degree of separation between peers. In our experiment, we use a random sample of certain percent of the graph nodes to compute *APL*.

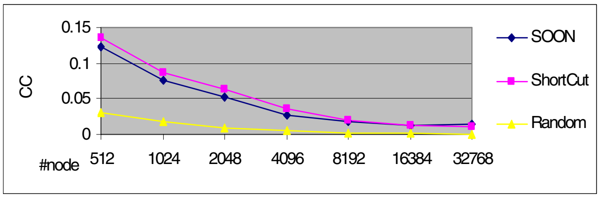


Fig. 3. Comparison of clustering coefficient

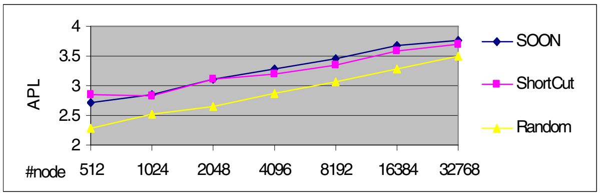


Fig. 4. Comparison of average path length

We performed experiments to measure SOON's *CC* and *APL*. An interest-based ShortCut topology and a random power-law topology with the same average node degree are used as reference topologies. The former has been proved to be a small-world network [15]. Fig.3 and Fig.4 show plots of the *CC* and the *APL* as a function of the number of nodes in the network. We observe that both the *CC* and the *APL* of SOON are very similar to those of ShortCut. The *CC* of SOON and ShortCut are much larger than that of the random power-law network, while the *APL* of SOON and ShortCut are almost the same as that of the random network. This indicates the emergence of a small-world network topology [16].

Scalability and efficiency

We examine the system performance in three different aspects, namely routing scalability, efficiency, and accuracy. The performance is measured using the metric of recall rate, which is defined as the number of results returned divided by the number of results actually available in the network. First, we vary the number of nodes from 2^9 to 2^{15} to test the scalability. The results are listed in Fig.5. As we expected, SOON gets higher recall in all these different sized networks. In addition, SOON's recall decreases less with the increase in network size. Fig.6 illustrates the system efficiency by showing the relationship between query recall rate and query TTL. With a small TTL, SOON gets a higher recall rate than the other two network. This means that SOON resolves queries faster than the others. In Fig.7 we show the effect of dispatching a different number of walkers to search the network. We can see that with the same TTL, SOON locates more results with fewer walkers.

SOON's small-world topology effectively reduces the search space, and its ontology summary guides the query in the right direction. This explains why SOON scales to large network size and why it achieves higher recall with shorter TTL and fewer walkers. Besides all these reasons, another factor contributing SOON's overall better recall rate is that SOON is able to locate semantically related results that cannot be located by the ShortCut and random-walk. Because of the semantic heterogeneity of our experimental setup, relevant resources may be represented with different ontologies. SOON may use its ontology signature set to find semantically related nodes and use the mapping defined to translate the query. Therefore, it can locate most of the relevant results. However, for ShortCut and random-walk, they have no way to find semantically related resources, but only resources represented in the same ontology as the ontology of the querying node.

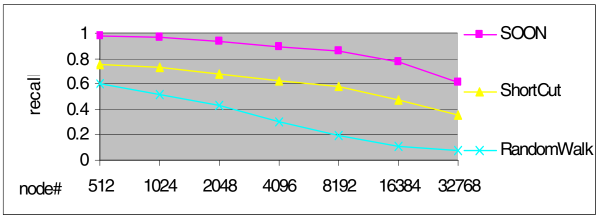


Fig. 5. Comparison of average path length

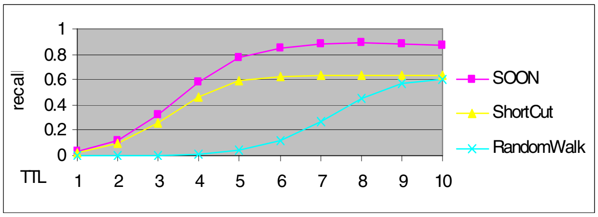


Fig. 6. Comparison of average path length

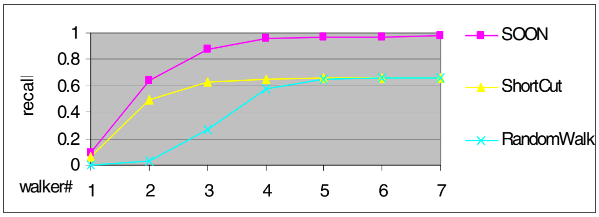


Fig. 7. Comparison of average path length

Overhead and adaptability to dynamics

The good recall performance of SOON does not come for free. Generally speaking, there is a tradeoff between query efficiency and maintenance overhead. Unlike ShortCut and random-walk approaches, which only create query propagating overhead, SOON also creates overhead for maintaining neighborhood relationship. We expect the extra overhead is reasonable and the saving from query cost exceeds the extra maintenance cost. To verify this, we examine the system's overhead in terms of accumulated bandwidth. System overhead has a close relation with the system dynamics, as a system must maintain consistent information about peers in the system in order to operate most effectively. Therefore, we measure the system dynamics together with the overhead. To evaluate the adaptability to different levels of dynamics, we measure the system overhead under different levels of peer "churn rate", referring to the rate of peers leaving/joining the system.

The experiment shown in Fig.8 gives an overview of how dynamics affect the system performance. We find that SOON performs similarly to the ShortCut algorithm which is proved to be resilient to churn [11]. When peers join or leave frequently, the performance of ShortCut and SOON deteriorate gracefully. Churn does not affect the two schemes dramatically because both algorithms do not depend on a strict structure to perform routing as DHTs do. Their unstructured random topologies provide multiple routes to a destination thus increasing the system resilience. In the worst case, they degrade to random-walk.

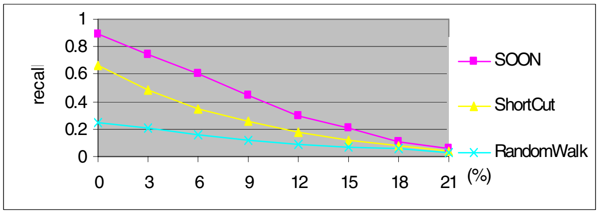


Fig. 8. Recall vs. churn rate

Fig.9 shows the accumulated bandwidth overhead of finding 10000 results under different churn rates. From the figure, we can see that in most situations SOON produces much less overhead than the other two methods. But when the system is very dynamic, such as when the churn rate is beyond 20%, SOON produces much more overhead. The high overhead problem of SOON in very dynamic environments

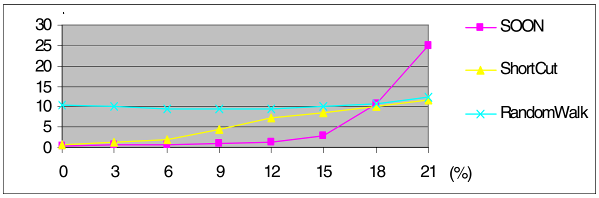


Fig. 9. System overhead (accumulated bandwidth) vs. churn rate

can be solved by a simple solution: when the network is very dynamic, the system can give up the ontology-based topology construction but resort to basic Gnutella random-walk as the solution.

6 Related Work

Research has harnessed the power of semantic technologies to aid in information representation, retrieval and aggregation over large distributed systems. P2P technology has been used to improve the scalability and efficiency of the semantic searching. For example, systems such as Edutella [17] and InfoQuilt [19] use broadcast or flooding to search their semantic metadata, while many other projects, such as RDFPeer [20] and OntoGrid [21] attempt applying DHT techniques to the retrieval of the ontology encoded knowledge. pSearch [22] applies a dimension reduction technique, called rolling index, on top of CAN to realize a semantics-based search.

Recently, there has appeared the idea of grouping nodes with similar contents together to facilitate search. The latest super-peer-based Edutella [18] and Semantic Overlay Network (SON) [23] rely on centralized server or super-peers to cluster contents and nodes. Semantic Small Word (SSW) position peers and data objects in the semantic space, so that peers with similar data objects form into clusters. It then applies a dimension reduction technique on top of the DHT to realize a semantics-based search. In SSW, semantics of data objects is represented by a multi-attribute vector, but not Semantic Web-based data. Applications such as REMINDIN [11], Helios[13], and Bibster [24] add semantic short-cuts to group nodes. The short-cut approach relies on the presence of interest-based locality. Each peer builds a shortcut list of nodes that answered previous queries. To find content, a peer first queries the nodes on its shortcut list and only if unsuccessful, floods the query.

7 Conclusion

In this paper, we propose a self-organized semantic overlay network, SOON. It uses an ontology-based representation of the information metadata. It enables peers to automatically organize themselves according to their semantic properties to form a semantic small-world topology, so that information retrieval can be effectively performed within semantically related small-worlds. Our simulation results prove that SOON improves interoperability among network participants and aids efficient information discovery and access.

References

1. Barabási, A.L.: Linked: How Everything is Connected to Everything Else and What It Means for Business, Science, and Everyday Life. Efficient information discovery and access. Plume, New York (2003)
2. Milgram, S.: The small world problem. Psychology Today 67(1) (1967)
3. Kleinberg, J.: Navigation in a small world. Nature (406), 845 (2000)
4. Gruver, W.A., Boudreaux, J.C.: Intelligent Manufacturing: programming environments for CIM. Springer, London (1993)
5. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.J.: Introduction to WordNet: an on-line lexical database. International Journal of Lexicography (1990)
6. Jiang, J., Conrath, D.: Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In: Proc. Int'l Conf. Computational Linguistics (ROCLING X) (1997)
7. Lee, J., Kim, M., Lee, Y.: Information Retrieval Based on Conceptual Distance in IS-A Hierarchies. J. Documentation 49, 188–207 (1993)
8. Rodriguez, M.A., Egenhofer, M.J.: Determining Semantic Similarity Among Entity Classes from Different Ontologies. IEEE Transactions on Knowledge and Data Engineering 15(2) (March/April 2003)
9. Tversky, A.: Features of similarity. Psychological Review 84(4), 327–352 (1977)
10. Lin, D.: An information-theoretic definition of similarity. In: Proc. 15th International Conf. on Machine Learning, pp. 296–304. Morgan Kaufmann, San Francisco (1998)
11. Tempich, X., Staab, S., Wranik, A.: REMINDIN: semantic query routing in peer-to-peer networks based on social metaphors. In: International World Wide Web Conference (WWW), New York, USA (2004)
12. Sripanidkulchai, K., Maggs, B., Zhang, H.: Efficient content location using interest-based locality in peer-to-peer systems. In: INFOCOM 2003 (2003)
13. Castano, S., Ferrara, A., Montanelli, S., Zucchelli, D.: Helios: a general framework for ontology-based knowledge sharing and evolution in P2P systems. In: IEEE Proc. of DEXA WEBS 2003 Workshop, Prague, Czech Republic (September 2003)
14. Castano, A., Ferrara, S., Montanelli, S., Pagani, E., Rossi, G.: Ontology addressable contents in p2p networks. In: Proceedings of the WWW 2003 Workshop on Semantics in Peer-to-Peer and Grid Computing (2003)
15. Iamnitchi, A., Ripeanu, M., Foster, I.: Small-world filesharing communities. In: Infocom, Hong Kong, China (2004)
16. Watts, D., Strogatz, S.: Collective dynamics of "small-world" networks. Nature (1998)
17. Nejdl, W., Wolf, B., Qu, C., Decker, S., SIntek, M., Naeve, A., Nilsson, M., Palmer, M., Risch, T.: Edutella: A P2P Networking Infrastructure Based on RDF. In: WWW 2002, Honolulu, Hawaii, USA, May 7-11 (2002)
18. Nejdl, W., Siberski, W., Sintek, M.: Design Issues and Challenges for RDF an schema-based peer-to-peer systems. ACM SIGMOD Record 32(3), 41–46 (2003)
19. Arumugam, M., Sheth, A., Arpinar, I.B.: Towards peer-to-peer semantic web: A distribuited environment for sharing semantic knowledge on the web. In: Proc. of the International World Wide Web Conference 2002 (WWW 2002), Honolulu, Hawaii, USA (2002)
20. Cai, M., Frank, M.: RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In: Proc. of WWW conference, NewYork, USA (May 2004)
21. OntoGrid project: `http://www.ontogrid.net/`

22. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks, In: Proceedings of 2003 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (2003)
23. Crespo, A., Garcia-Molina, H.: Semantic overlay networks. Technical report, Stanford University (2002)
24. Castano, A., Ferrara, S., Montanelli, S., Pagani, E., Rossi, G.: Ontology addressable contents in p2p networks. In: Proceedings of the WWW 2003 Workshop on Semantics in Peer-to-Peer and Grid Computing (2003)
25. Gnutella website, `http://gnutella.wego.com/`
26. College of American Pathologists. SNOMED RT - Systematized Nomenclature of Medicine Reference Terminology, VERSION 1.1, USER GUIDE (2001)
27. Li, J., Vuong, S.: An Ontological Framework for Large-Scale Grid Resource Discovery. In: Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2007), Aveiro, Portugal (July 2007)

Dynamic Overlay Node Activation Algorithms for Large-Scale Service Deployments

Jeroen Famaey*, Tim Wauters**, Filip De Turck**, Bart Dhoedt, and Piet Demeester

Department of Information Technology (INTEC), Ghent University - IBBT
Gaston Crommenlaan 8, bus 201, B-9050 Gent, Belgium
jeroen.famaey@intec.ugent.be

Abstract. Due to overprovisioning of infrastructure nodes in overlay networks, many nodes remain idle at times of low network load. Some of these nodes could be temporary removed from the overlay topology and could then be used for other purposes or alternatively be temporarily shut down, to save energy.

In this paper we present several algorithms to select the subset of overlay nodes that should be part of the overlay topology. This decision is made based on the current and (estimated) future network load and the locality of the clients and servers connected to the overlay network. As network load or conditions change, additional nodes can be dynamically (de)activated. Our algorithms can be used in conjunction with existing overlay topology construction protocols.

Through extensive simulations, we have evaluated and compared the performance of our algorithms.

1 Introduction

Recently, peer-to-peer overlay networks have been increasingly used to offer services on the Internet, such as Voice-over-IP (e.g. Skype), filesharing (e.g. Gnutella, BitTorrent), and Internet TV (e.g. Joost, Zattoo).

Although many protocols have been proposed to dynamically create and manage both structured and unstructured overlay topologies in a scalable manner, little attention has been paid to deciding which candidate overlay nodes to actually include in the overlay network. Existing protocols merely include every available candidate without any sort of node activation mechanism. As a consequence, many nodes in overprovisioned overlay networks remain idle. By adding a node activation phase to the overlay construction mechanism, the resources of these idle overlay nodes can be freed. They could be used for other purposes or be temporarily shut down to save energy. Additionally, if included in the overlay, these idle nodes take part in control and management protocols and algorithms

* Funded by the Institute for the Promotion of Innovation by Science and Technology Flanders (IWT-Vlaanderen).

** Funded by the Fund for Scientific Research Flanders (FWO-Vlaanderen).

run on top of the overlay network, adding extra control-traffic overhead and negatively influencing overall performance and scalability [1].

In this paper, we propose two heuristics to select a suitable subset of the available overlay nodes, to be used in the overlay topology. The first heuristic is static and centralized, suitable for determining where to place the initial overlay nodes. The second heuristic is fully decentralized and thus suitable for managing large-scale topologies. Additionally, it is capable of dynamically adjusting the subset of activated overlay nodes. To compare the solution of these heuristics to the global optimal solution, we have also implemented an optimal algorithm to solve the problem, based on an Integer Linear Programming (ILP) formulation.

The two heuristics are specifically designed for service hosting overlays. In this case the overlay network consists not only of overlay nodes, which are responsible for routing client requests and selecting services, but also of clients and service hosts. As input, the activation algorithms take into account the demand generated by nearby clients, the demand satisfied by nearby service hosts, and the bandwidth load on incoming and outgoing overlay links. As a result the algorithms return which overlay nodes should be activated but also which overlay node to use as the gateway for each client and service host. The goal is to make sure there are enough resources available in the network to satisfy the client demand, while minimizing the amount of activated overlay nodes.

The rest of this paper is structured as follows. Related work is discussed in Section 2. The node activation and gateway selection problems are formally described in Section 3. Section 4 gives a description of the designed algorithms. An evaluation based on simulation results is presented in Section 5. And finally, a brief summary of our work is given in Section 6.

2 Related Work

In the context of overlay service hosting platforms, most research has been aimed towards service placement and server selection algorithms [2,3], while problems related to overlay construction have received little attention. In the context of general-purpose overlay networks, most research has been done towards constructing overlay topologies, both structured [4] and unstructured [5,6,7]. These protocols include every available overlay node in the topology, instead of selecting a subset of required nodes, based on the current network load. Our node activation algorithms can be used in conjunction with these overlay topology construction protocols, to improve their resource usage, and the performance of the resulting overlay network.

As mentioned in Section 1, the overlay node activation problem has received little attention so far. As far as we are aware, the problem has only been studied in the context of intrusion-tolerant networks [1]. The authors propose an algorithm that selects overlay nodes from a set of inactive ones to replace unresponsive or compromised nodes. They, however, focus on security and Byzantine fault tolerance. In contrast, our main goal is to maximize the satisfied client demand, while minimizing the amount of idle resources.

3 Problem Formulation

Given is an underlying network topology represented by a bidirectional graph $U(N, E)$, with N a set of network nodes (e.g. clients, servers or routers) and E a set of edges connecting these nodes. Each edge e has a distance d_e (e.g. one-way transmission delay) and available bandwidth b_e.

Also given is a set of candidate overlay nodes $O \subseteq N$, a set of clients $C \subseteq N$ and a set of service-hosts $S \subseteq N$. Any two overlay nodes, clients or service-hosts m and n are connected through a path of edges $p_{m,n} \subseteq E$ (called an overlay edge) in the underlaying network (in our implementation the shortest hop-count path). The distance $d_{m,n}$ and bandwidth $b_{m,n}$ between two overlay nodes m and n are respectively defined as $\sum_{e \in p_{m,n}} d_e$ and $\min_{e \in p_{m,n}} b_e$.

Every overlay node $o \in O$ has a limited amount of resources Ω_o and a set of nearby overlay nodes $N_o \subset O$ (the neighbours of o). The neighbourhood set N_o of $o \in O$ contains all overlay nodes $v \in O$ for which $d_{o,v} \leq D$ (with D a predefined distance bound).

Every client and service-host $u \in C \cup S$ has an amount of required resources ω_u (e.g. CPU, memory), which represent the resources needed by the gateway (the overlay node used to access the overlay network) to perform overlay-related tasks. For clients this would consist of service discovery, server selection and request routing. For service-hosts this would be service placement, service advertisement and reply routing.

The set M_o denotes all $u \in C \cup S$ that have $o \in O$ as nearest overlay node. This set contains all clients and service-hosts for which $d_{u,o} = \min_{q \in O} d_{u,q}$.

Additionally we define a set of available services T. Each client $c \in C$ has requests for a subset of these services $T_c \subseteq T$. The variable $r_{c,t}$ denotes the amount of requests per second of client $c \in C$ for service $t \in T_c$. Each service-host $s \in S$ runs a subset of all available services. Each service $t \in T$ has a required amount of bandwidth β_t^o per request and β_t^i per reply. The service-host that satisfies the demand of client $c \in C$ for service $t \in T_c$ is denoted by $s_{c,t}$.

The goal of overlay node activation is to select a subset $A \subseteq O$ of active overlay nodes. Additionally a gateway $o \in A$ should be selected for each client and service-host $u \in C \cup S$. Several constraints should be satisfied during the activation process. These can be summarized as follows

$$\forall o \in A : \sum_{u \in U_o} \omega_u \leq \Omega_o \ . \tag{1}$$

$$\forall o \in A, \forall u \in U_o : d_{u,o} \leq K \ . \tag{2}$$

$$\forall e \in E : \sum_{c \in C} \sum_{t \in T_c} \left(\sum_{p \in \pi^o_{c,t} \wedge e \in p} r_{c,t} \cdot \beta_t^o + \sum_{p \in \pi^i_{c,t} \wedge e \in p} r_{c,t} \times \beta_t^i \right) \leq b_e \ . \tag{3}$$

With U_o the set of clients and service-hosts using $o \in A$ as gateway, K the predefined maximum allowed distance between a client or service-host and their gateway, and $\pi^o_{c,t}$ and $\pi^i_{c,t}$ respectively the set of overlay edges in the overlay path from the client c to the service-host $s_{c,t}$ and back. Eq. 1 dictates that the resources used by all clients and service-hosts using an overlay node as a gateway should not exceed the available resources of that overlay node. Eq. 2 states that the distance between a client or service-host and its gateway should not exceed a predefined distance bound K. And finally Eq. 3 stipulates that the bandwidth used on each underlay edge to send requests and replies for all clients should not exceed the available bandwidth of that edge.

The optimization goal consists of three subgoals, which can be described as follows

1. Satisfied client demand should be maximized. This means that enough bandwidth should be available in the overlay network to send as many service requests and replies between clients and service-hosts as possible.
2. A gateway should be provided for as many clients and service-hosts as possible.
3. Enough overlay nodes should be activated in order to maximize the first two goals, but not more. This goal minimizes the amount of idle overlay nodes.

4 Algorithms Description

4.1 Integer Linear Programming Approach (ILP)

Based on the formal problem description given in Section 3 we can define an ILP formulation for the problem. This formulation can be used to find the optimal solution for the overlay node activation problem. Although this approach scales very poorly, it is still useful for comparing the solution of the heuristics to the optimal.

We will start by defining several decision variables used by the constraints and objective function

- a_o is 1 if $o \in A$ (o is activated) and 0 otherwise
- $g_{u,o}$ is 1 if $u \in U_o$ ($o \in A$ is the gateway of $u \in C \cup S$) and 0 otherwise
- $q^{o/i}_{c,t,m,n}$ is 1 if $p_{m,n} \in \pi^{o/i}_{c,t}$ (with $m, n \in O \cup C \cup S$) and 0 otherwise
- $r^{o/i}_{c,t,m,n}$ is the amount of satisfied requests of $c \in C$ for $t \in T_c$ on $p_{m,n} \in \pi^{o/i}_{c,t}$

The ILP formulation consists of several constraints, which limit the possible values of the decision variables to valid solutions only. The first three constraints are the same as the ones given in Section 3 (though formulated using the decision variables). Additionally, there are several other constraints. These include constraints that make sure every client has only 1 gateway and that all gateways and overlay nodes used for routing client requests or replies are activated. Finally, there are also several flow conservation constraints.

The optimization goal of maximizing the total satisfied demand and number of clients and service-hosts with a gateway while minimizing the total number of activated overlay nodes, can be translated into the following ILP objective function

$$\max \sum_{o \in O} \left(\sum_{u \in C \cup S} 2 \cdot \omega_u \cdot g_{u,o} \right) + \left(\sum_{c \in C} \sum_{t \in T_c} \left(\beta^o_t + \beta^i_t \right) \cdot r^o_{c,t,c,o} \right) - a_o \ .$$

Algorithm 1. The static node activation heuristic (STANA)

```
procedure SelectGateways()
1: for all $o \in O$ do
2:     for all $c \in C_o$ do
3:         $n \leftarrow$ o.GetMostSatisfyingGateway($c$); $C_o \leftarrow C_o \setminus \{c\}$; $C_n \leftarrow C_n \cup \{c\}$
4: for all $o \in O$ do
5:     for all $s \in S_o$ do
6:         $n \leftarrow$ o.GetMostSatisfyingGateway($s$); $S_o \leftarrow S_o \setminus \{s\}$; $S_n \leftarrow S_n \cup \{s\}$

$b \in O$ function o.GetMostSatisfyingGateway($u$)
1: $b \leftarrow o$; $r \leftarrow$ o.GetSatisfiableDemand($u$)
2: for all $n \in O$ do
3:     if $d_{u,n} \leq K$ and $\omega_u + \sum_{v \in C_n \cup S_n} \omega_v \leq \Omega_n$ then
4:         if $\sum_{v \in C_o \cup S_o} \omega_v < \sum_{v \in C_n \cup S_n} \omega_v$ or $\sum_{v \in C_o \cup S_o} \omega_v > \Omega_o$ then
5:             if n.GetSatisfiableDemand($u$) $> r$ then
6:                 $b \leftarrow n$; $r \leftarrow$ n.GetSatisfiableDemand($u$)
7: return $b$
```

An algorithm to solve the ILP formulation was implemented using ILOG CPLEX [8], which solves ILP problems using the simplex and interior point methods.

4.2 Static Node Activation (STANA)

The pseudocode of the heuristic is shown in Algorithm 1. The heuristic is initiated by calling the `SelectGateways()` procedure. First it attempts to find the best gateway for each client (lines 1-3). Then it attempts to find the best gateway for the service-hosts (lines 4-6). C_o and S_o represent the clients and service-hosts that have o as their gateway. They are initialized to the clients and service-hosts in M_o.

The `GetMostSatisfyingGateway()` function returns an overlay node b for the given client or service-host u. The number of requests of u that can be satisfied is maximal if b is the gateway of u. The function iterates over all overlay nodes (line 2). First it checks if the overlay node n meets the requirements to be the gateway of u (line 3). The distance from u to n should be less than K and n should have enough free resources (e.g. CPU). Subsequently the function checks if n has a higher load than the current gateway o of u or if o is overloaded (line 4). The reason an overlay node will only transfer clients and service-hosts to neighbours with higher load (unless it is overloaded) is based on the observation that it would be easier to transfer all clients and service-hosts away from a node with only few of them, than from a node with many. This allows the heuristic to empty as many nodes with low load as possible and minimize the number of overlay nodes that should be activated. Finally the function checks if n would be a better gateway than the current best gateway b and replaces it if necessary (lines 5-6).

After calling the procedure all overlay nodes o for which $C_o \cup S_o \neq \emptyset$ will be activated. All overlay nodes on the path from a client c to one of its service-hosts $s_{c,t}$ or back will also be activated.

4.3 Dynamic Node Activation (DYNNA)

Although the static heuristic is interesting from a theoretical point of view and for initial network dimensioning, it cannot adapt to changing conditions (such as changes in client demand, underlying network conditions, or node churn). To be able to quickly adapt to these changes we have devised a dynamic heuristic. Additionally, this heuristic runs fully distributed and overlay nodes use only measurable information from their direct neighbours (current resource load, overlay edge transmission delay and overlay edge bandwidth usage). In contrast, the static heuristic requires a view on the entire overlay network and needs information on expected future bandwidth usage of each client and which service-hosts each client will use.

Pseudocode for the dynamic heuristic is shown in Algorithm 2. Several new variables are introduced: specifically Ω_{min}, Ω_{max}, β_{min}, β_{max}, $b^u_{m,n}$, b^{out}_u and b^{in}_u. Ω_{min} and Ω_{max} define the minimum and maximum resource load percentage of an overlay nodes before it should attempt to transfer clients and service-hosts to other overlay nodes. β_{min} and β_{max} have simular meanings, but for the overlay edge bandwidth. $b^u_{m,n}$ denotes the used bandwidth on the overlay edge between any two overlay nodes, clients, or service-hosts m and n. Finally b^{out}_u and b^{in}_u denote the bandwidth currently provided to client u, respectively to and from its gateway.

A node initiates the algorithm by calling the `ReDistribute()` procedure. This could be done at fixed intervals or when the node detects a significant change in resource load or bandwidth usage of its overlay edges. The heuristic consists of 5 phases. The first phase (lines 2-8) is only initiated if the resource load percentage of the overlay node is greater than Ω_{max}, meaning it is either overloaded or nearly overloaded. It will first attempt to transfer clients and service-hosts to its active neighbours (lines 3-4). If this fails it will activate additional neighbours and try to transfer clients and service-hosts to them (lines 5-8). The second phase (lines 9-11) is only initiated if the overlay node has a resource load of Ω_{min} or less and the bandwidth usage of its outgoing links is less than β_{min}. This means the node has very low load, and only few service requests or replies pass by it. The node will attempt to transfer all its clients and service-hosts to active neighbours, so it can deactivate itself. In the third phase (lines 12-16) the node checks the available bandwidth on the links to and from its connected clients and service-hosts (lines 12-14). All clients and service-hosts that might require more bandwidth are, when possible, transferred to other nodes with more available bandwidth (lines 15-16). In the fourth phase (lines 17-18) the node checks if the average bandwidth usage on its outgoing overlay edges exceeds β_{max}. If it does, the node attempts to increase the available bandwidth in the overlay by activating extra neighbours. In the fifth and final phase (lines 19-20) the overlay

Algorithm 2. The dynamic node activation heuristic (DYNNA)

```
procedure o.ReDistribute()
 1: $T \leftarrow \{\}$ ; $A \leftarrow$ GetActive($N_o$) ; $I \leftarrow N_o \setminus A$ ; $U_o \leftarrow C_o \cup S_o$
 2: if $\sum_{u \in U_o} \omega_u > \Omega_{max} \cdot \Omega_o$ then
 3:     while HasNext($A$) and $\sum_{u \in U_o} \omega_u > \Omega_{max} \cdot \Omega_o$ do
 4:         $s \leftarrow$ GetNext($A$) ; $U_o \leftarrow U_o \setminus$ s.TransferNodes($U_o$, $\{o\}$)
 5:     while HasNext($I$) and $\sum_{u \in U_o} \omega_u > \Omega_{max} \cdot \Omega_o$ do
 6:         $s \leftarrow$ GetNext($I$) ; s.Activate() ; $A \leftarrow A \cup \{s\}$ ; $I \leftarrow I \setminus \{s\}$ ;
 7:         $U_o \leftarrow U_o \setminus$ s.TransferNodes($U_o$, $\{o\}$)
 8:     $U_o \leftarrow U_o \setminus$ o.GetOverloadingNodes($U_o$)
 9: if $\sum_{u \in U_o} \omega_u < \Omega_{min} \cdot \Omega_o$ and $\mathrm{avg}_{s \in A} \frac{b^u_{s,o}}{b_{s,o}} < \beta_{min}$ then
10:     while HasNext($A$) and $|U_o| > 0$ do
11:         $s \leftarrow$ GetNext($A$) ; $U_o \leftarrow U_o \setminus$ s.TransferNodes($U_o$, $\{o\}$)
12: for all $u \in C_o \cup S_o$ do
13:     if $b^u_{u,o} > \beta_{max} \cdot b_{u,o}$ or $b^u_{o,u} > \beta_{max} \cdot b_{o,u}$ then
14:         $T \leftarrow T \cup \{u\}$
15: for all $s \in A$ do
16:     $U_o \leftarrow U_o \setminus$ s.TransferNodes($T \cap U_o$, $\{o\}$)
17: while HasNext($I$) and $\mathrm{avg}_{s \in A} \frac{b^u_{o,s}}{b_{o,s}} > \beta_{max}$ do
18:     $s \leftarrow$ GetNext($I$) ; s.Activate() ; $A \leftarrow A \cup \{s\}$ ; $I \leftarrow I \setminus \{s\}$
19: if $|U_o| = 0$ and $\mathrm{avg}_{s \in A} \frac{b^u_{s,o}}{b_{s,o}} < \beta_{min}$ then
20:     o.DeActivate()

$T \subseteq U$ function o.TransferNodes($U$, $V$)
 1: $T \leftarrow \{\}$ ; $A \leftarrow$ GetActive($N_o$) ; $W \leftarrow$ o.GetNodesInReach($U$) ; $V \leftarrow V \cup \{o\}$
 2: if $|W| > 0$ then
 3:     for all $u \in W$ do
 4:         if $\omega_u + \sum_{v \in U_o} \omega_v < \Omega_{max} \cdot \Omega_o$ then
 5:             if $b^{out}_u \leq \beta_{max} \cdot b_{u,o}$ and $b^{in}_u \leq \beta_{max} \cdot b_{o,u}$ then
 6:                 $U_o \leftarrow U_o \cup \{u\}$ ; $T \leftarrow T \cup \{u\}$
 7:     for all $s \in A \setminus V$ do
 8:         $T \leftarrow T \cup$ s.TransferNodes($W \setminus T$, $V$)
 9: return $T$
```

node deactivates itself if no clients or service-hosts use it as gateway and the incoming bandwidth on its overlay edges is below β_{min}.

The `TransferNodes()` function recursively attempts to find a new gateway for the clients and service-hosts in U. It returns a set T of all elements of U for which a new gateway was found. A node first checks if it can act as a gateway for any of the nodes in U (lines 3-6). Then it sends the remaining nodes to its active neighbours (lines 7-8) who recursively do the same. The extra V parameter is used to prevent loops. Nodes already visited in a recursive call will be stored in V and are not visited again. `GetNodesInReach()` merely returns all clients and service-hosts $u \in U$ for which $d_{u,o} \leq K$.

Note that the algorithm disconnects clients and service-hosts if a node is overloaded and no replacement gateway is found (line 8 of `ReDistribute()`).

Additionally, the algorithm does not check if the distance to the current gateway has become larger than K (it will however check this when looking for a new gateway). If a client or service-host has been disconnected or the distance to its gateway has changed it is itself responsible to find a new gateway (by reconnecting to another overlay node).

The join procedure for clients and service-hosts is described as follows. First it contacts a random overlay node (most likely selected from a set of static bootstrap servers) and requests a set of nearby overlay nodes. It then contacts the nearest from this set of nearby overlay nodes. If that node is active and it has enough free resources it will become the new gateway of the client (or service-host). If it is inactive or it does not have enough free resources it will attempt to find another gateway using the `TransferNodes()` function. If this is unsuccesful the node will activate itself if it was inactive or disconnect the client if it was already active.

5 Evaluation Results

In this section the performance, scalability and adaptability of the heuristics is discussed based on simulation results. Performance is measured by comparing the percentage of satisfied demand and activated overlay nodes to the optimal solution (calculated using the ILP algorithm) for growing bandwidth load. Scalability is evaluated by comparing results for a growing amount of available overlay nodes. Finally, adaptability is studied by dynamically changing the demand pattern. Satisfied demand is measured as the percentage of requests from clients that can be processed by service-hosts.

For reference purposes, we also implemented an underlay algorithm (UND). This algorithm routes data from the sender to the receiver directly via the shortest-hop-count path in the underlay network.

Statistical analysis was used to interpret simulation results. A one-way ANOVA [9] was used to compare several levels of a single factor. If an effect of the factor was detected, a Tukey test [9] was performed to determine which averages actually differed significantly. The 'homogeneity of variance' prerequisite for ANOVA was checked using a modified Levene test [9]. All statistics were performed using a 5% signficance level.

5.1 Simulation Setup

Random underlay networks were generated using the BRITE topology generator [10] with the hierarchical top-down model and Waxman's algorithm. The bandwidth of the links between underlay routers was chosen according to a uniform distribution in the interval $\{5, 15\}$. Each router had on average an outdegree of 2. All routers were placed on a 200×200 grid and the link delay was chosen directly proportional to the euclidean distance in this grid. A subset of all available routers was selected as access routers, all others are considered core routers. Clients and service-hosts were connected to a randomly selected access router by a fast and high-bandwidth link (0 ms delay and 100 Mbps bandwidth). Overlay nodes were directly connected to the access routers and high-degree core routers, also by fast, high-bandwidth links. The CPU resources of each overlay node were chosen uniformly from the interval $\{6, 8\}$. As the resource requirements of each client and service-host were set to 1, this means every overlay node could be the gateway of at most 6 to 8 devices.

5.2 Performance Results

The goal of this simulation was to evaluate the performance of the heuristics in terms of satisfied demand and amount of activated overlay nodes for different bandwidth loads (Mbps). The bandwidth load is measured as the total bandwidth needed by all client requests.

Because of the exponential time-complexity of the ILP algorithm, this simulation was performed only on small networks. Each network contained 20 routers and 12 overlay nodes.

Fig. 1 shows the simulation results as a function of bandwidth load (Mbps) and for different values of K (maximum distance to the gateway). As expected, the satisfied demand is inversely proportional and the number of activated overlay nodes is directly proportional to the bandwidth load.

Statisticaly analysis showed that, in terms of satisfied demand, ILP did not perform significantly better than DYNNA and only significantly better than STANA for $K = 150$ at 20 Mbps and higher. For $K = 50$, ILP performed at most respectively 7 and 9% better than DYNNA and STANA. On the other hand, both heuristics performed up to 25% better than UND.

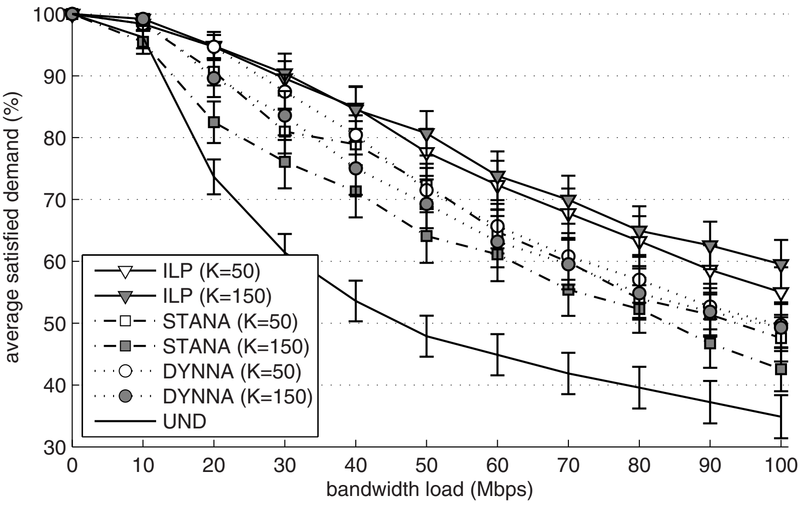
In terms of number of activated overlay nodes, ILP performed significantly better than both heuristics (respectively at most about 15% for $K = 50$ and 25% for $K = 150$). STANA performed significantly better than DYNNA only for $K = 150$, from 10 to 20 Mbps.

In summary, we can remark that the heuristics perform significantly better than classic underlay-based routing. There is however little difference between both heuristics. Additionally, for the heuristics there is little difference between results for different values of K.

5.3 Scalability Results

In this section, we study the scalability of the heuristics in terms of satisfied demand and amount of activated overlay nodes, for a growing number of candidate overlay nodes. As the ILP algorithm was not used for this simulation, larger test networks were possible. Tests were performed for networks with 2500 and 10000 underlay routers. The number of candidate overlay nodes was varied between 0 and 250.

Fig. 2 shows the simulation results as a function of available overlay nodes. As more overlay nodes become available, the amount of satisfied demand and number of used overlay nodes grows proportionally. It is striking that only a portion of available overlay nodes is used, even if only few are available. This is



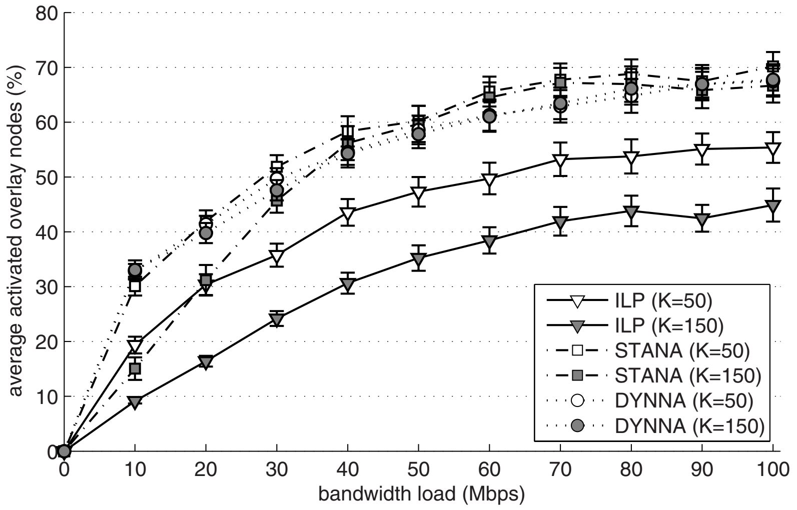
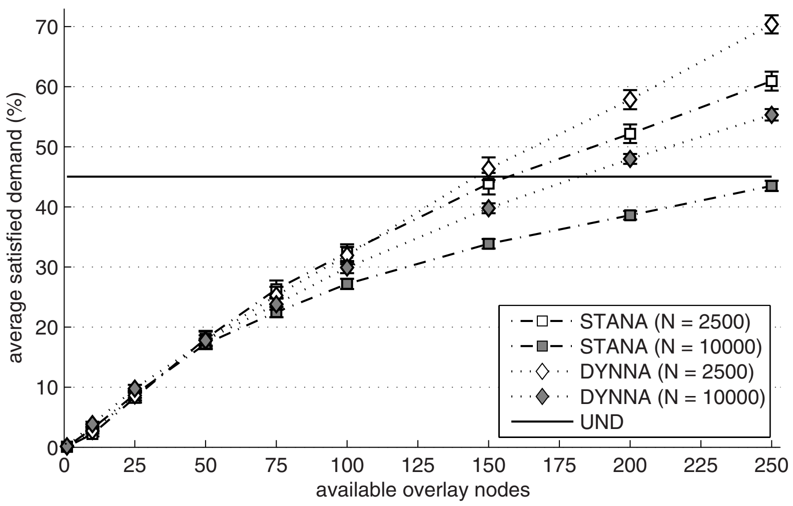


Fig. 1. The performance of the two heuristics (STANA, DYNNA) compared to the optimal algorithm (ILP) and underlay routing (UND) in terms of satisfied demand (%) (left) and activated overlay nodes (%) (right) for varying bandwidth load (Mbps) and maximum gateway distance K (averaged over 30 iterations - the error bars represent the standard error)

because the underlay paths corresponding to the overlay links of different overlay nodes might overlap. One of those nodes might cause the bandwidth in these links to become saturated. The algorithms will detect that there is no bandwidth available in the incoming or outgoing links of the other overlay node and will thus not use it.



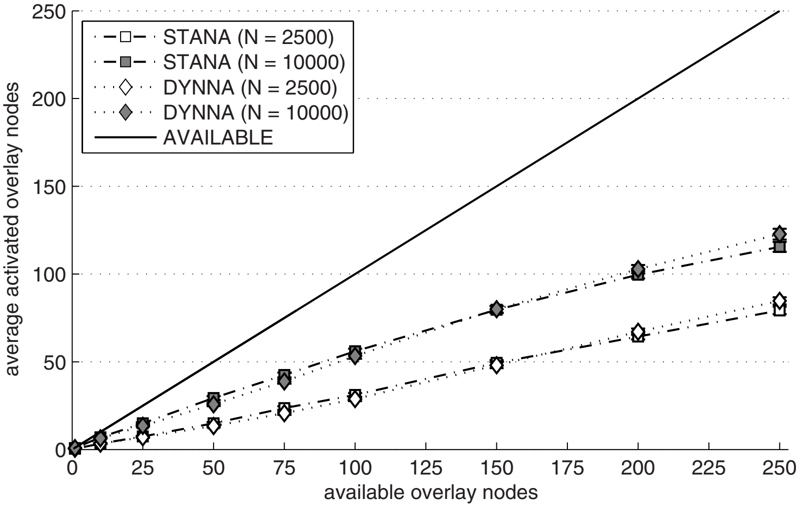
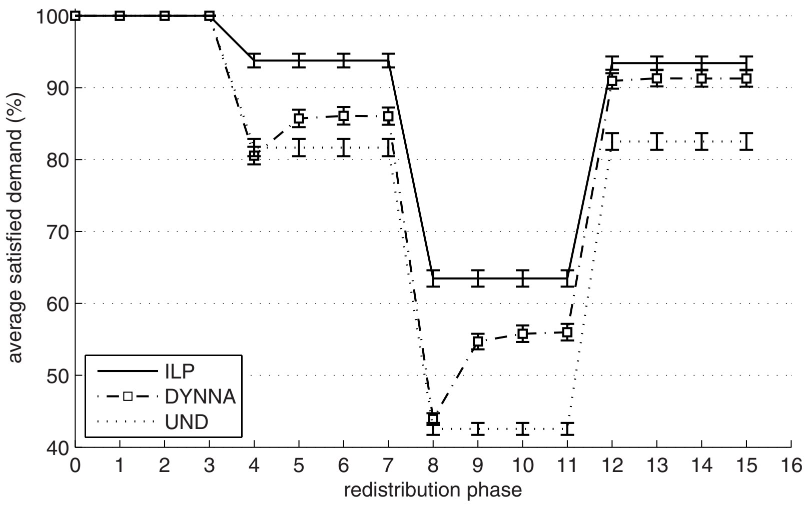


Fig. 2. The scalability of the two heuristics (STANA, DYNNA) compared to underlay routing (UND) in terms of satisfied demand (%) (left) and activated overlay nodes (%) (right) for varying number of available overlay nodes and for networks with 2500 and 10000 underlay routers (averaged over 39 iterations - the error bars represent the standard error)

Statistical analysis shows that DYNNA performs significantly better than STANA in terms of satisfied demand from 100 available overlay nodes and onwards when using 10000 routers, but only from 200 onwards when using 2500 routers. Additionally, no significant differences were found between both heuristics in terms of number of activated overlay nodes (for 2500 and 10000 routers).



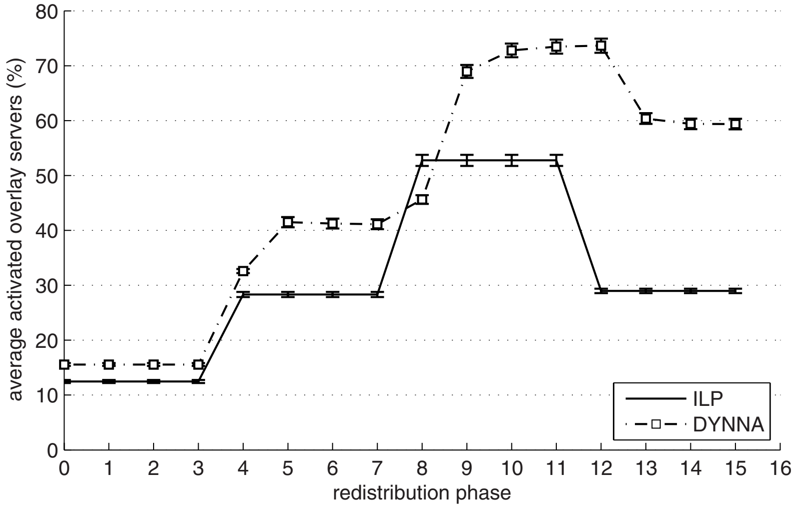


Fig. 3. The adaptability of DYNNA to dynamically changing demand compared to ILP and UND, in terms of satisfied demand (%) (left) and activated overlay nodes (%) (right) (averaged over 174 iterations - the error bars represent the standard error)

In summary, we have shown that, in line with the first experiment, there is little difference between both heuristics in terms of the amount of overlay nodes they activate. Additionally these simulations show that even when placing overlay nodes near hotspots (routers with high link degree) not all overlay nodes can be efficiently used, because of bandwidth constraints. Therefore, the presented node activation heuristics could prove to be very useful for deriving suitable locations for overlay infrastructure nodes.

5.4 Dynamic Demand Results

In this section, DYNNA is evaluated in face of dynamically changing demand patterns. Tests were performed on networks with 40 routers and 20 overlay nodes. In our example scenario, the demand was initialized to 0 Mbps and subsequently changed to 50, 200 and back to 50 Mbps. After each change in demand the dynamic algorithm's `ReDistribute()` was called 4 times on each node (called redistribution phases), to make sure the solution stabilized before the demand changed again. Consequently, demand increases occurred after phase 3 and 7 and a decrease after phase 11.

Fig. 3 shows the simulation results after subsequent redistribution phases. The figure shows that in terms of satisfied demand, one redistribution phase is needed for the results to stabilize after an increase in demand, and none after a decrease. On the other hand, the number of activated overlay nodes needs one redistribution phase to stabilize, both after an increase and decrease in demand. Additionally, it should be noted that after a decrease in demand (phase 11) the number of activated overlay nodes remains higher than it was before the identical increase (phase 7). As an interesting side effect, satisfied demand remains closer to optimal as well.

6 Summary

In this paper, we presented an optimal algorithm and two heuristics to solve the node activation problem in the context of service hosting platforms. The optimal algorithm (ILP) and the first heuristic (STANA) solve only a static version of the problem. This means that the request pattern for all clients and the entire network topology is known (or estimated) in advance. The second heuristic (DYNNA) is capable of dynamically (de)activating overlay nodes to cope with changes in demand patterns and client or service-host churn. To improve scalability it can also be run in a distributed way, as each overlay node only requires knowledge on its direct overlay neighbours and the links between them.

Using extensive simulations, we showed that the dynamic (distributed) heuristic performs as good, if not better, than the static heuristic. Additionally, we showed that when bandwidth is plentiful, the heuristics do not perform significantly worse than the optimal algorithm.

Overlay node activation mechanisms provide several advantages to any overlay-based platform. First, dynamic node activation allows idle resources to be temporarily freed and used for other purposes. Intelligent algorithms can reclaim or free nodes to cope with fluctuations in overlay-resource demand. Second, idle nodes no longer take part in overlay control and management protocols, improving overall performance and scalability. And finally, static node activation can be used to find suitable locations for infrastructure nodes.

References

1. Obelheiro, R.R., Fraga, J.S.: Overlay network topology reconfiguration in byzantine settings. In: IEEE Pacific Rim Dependable Computing Conference (PRDC 2007), pp. 155–162 (2007)
2. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: Proceedings of the 15th International Conference on World Wide Web (WWW 2006), pp. 595–604 (2006)
3. Adam, C., Stadler, R., Tang, C., Steinder, M., Spreitzer, M.: A service middleware that scales in system size and applications. In: 10th IFIP/IEEE International Symposium on Integrated Management (IM 2007), pp. 70–79 (2007)
4. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001), pp. 149–160 (2001)
5. Tang, C., Chang, R.N., Ward, C.: Gocast: Gossip-enhanced overlay multicast for fast and dependable group communication. In: Conference on Dependable Systems and Networks (DSN 2005), pp. 140–149 (2005)
6. Voulgaris, S., Gavidia, D., van Steen, M.: CYCLON: Inexpensive membership management for unstructured P2P overlays. Journal of Network and Systems Management 13(2), 197–217 (2005)
7. Ganesh, A.J., Kermarrec, A.M., Massoulie, L.: Peer-to-peer membership management for gossip-based protocols. Transactions on Computers 52(2), 139–149 (2003)
8. ILOG: CPLEX 10.0 User's Manual. ILOG Inc., Mountain View, CA (2006)
9. Hill, T., Lewicki, P.: Statistics: Methods and Applications. StatSoft, Inc. (2006)
10. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: An approach to universal topology generation. In: Proceedings of the Internation Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2001) (2001)

Dynamic Querying in Structured Peer-to-Peer Networks

Domenico Talia and Paolo Trunfio

DEIS, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy
{talia,trunfio}@deis.unical.it

Abstract. Dynamic Querying (DQ) is a technique adopted in unstructured Peer-to-Peer (P2P) networks to minimize the number of peers that is necessary to visit to reach the desired number of results. In this paper we introduce the use of the DQ technique in structured P2P networks. In particular, we present a P2P search algorithm, named DQ-DHT (Dynamic Querying over a Distributed Hash Table), to perform DQ-like searches over DHT-based overlays. The aim of DQ-DHT is two-fold: allowing arbitrary queries to be performed in structured P2P networks, and providing dynamic adaptation of the search according to the popularity of the resources to be located. This paper describes the DQ-DHT algorithm using Chord as basic overlay and analyzes its performance in comparison with DQ in unstructured networks.

1 Introduction

Structured Peer-to-Peer (P2P) systems like Chord [1] keep association of resource identifiers to nodes using a Distributed Hash Table (DHT), which allows to locate the node responsible for the resource with a given Id (or key) with logarithmic performance bounds. As compared to unstructured P2P systems like Gnutella [2], however, structured systems provide a limited support to complex queries. Although several extensions to basic DHT schemes have been proposed to support, for instance, range queries [3], multi-attribute search [4], and keyword-based search [5], DHT-based lookups still do not support arbitrary queries (e.g., regular expressions [6]) since it is infeasible to generate and store keys for every query expression. On the other hand, unstructured systems can do it effortless since all queries are processed locally on a node-by-node basis [7].

Even if the lookup mechanisms of DHT-based systems do not support arbitrary queries, it is possible to exploit their structure to distribute any kind of information across the overlay with minimal cost. For example, in [8] a technique for efficient broadcast over a DHT is proposed. Using such technique, a broadcast message originating at an arbitrary node in the DHT overlay reaches all other nodes without redundant messages in $O(\log N)$ steps. It can be used to broadcast arbitrary types of queries, which can be then processed locally by single nodes as in unstructured systems. We elaborate on such an approach by proposing a P2P search algorithm, named DQ-DHT (Dynamic Querying over a Distributed Hash Table), to provide efficient execution of arbitrary queries in structured P2P networks. DQ-DHT is based on a combination of the broadcast technique mentioned above with the Dynamic Querying (DQ) technique [9] used in unstructured networks.

The goal of DQ is to minimize the number of nodes that is necessary to visit in an unstructured network to obtain the desired number of results. The query initiator starts the search by sending the query to a few of its neighbors and with a small Time-To-Live (TTL). The main goal of this "probe" query is to estimate the popularity of the resource to be located. If such an attempt does not produce a sufficient number of results, the search initiator sends the query towards the next neighbor with a new TTL. Such TTL is calculated taking into account both the desired number of results and the resource popularity estimated during the previous phase. This process is repeated until the expected number of results is received, or there are no more neighbors to query.

Similarly to DQ, DQ-DHT performs the broadcast in an iterative way until the target number of results is obtained. At each iteration, a new subset of nodes is queried on the basis of the estimated resource popularity and the desired number of results. Differently from DQ, DQ-DHT exploits the structural constraints of the DHT to avoid message duplications and ensure higher success rate.

DQ-DHT has been particularly designed to serve as resource discovery mechanism for decentralized infrastructures like computational Grids. Large-scale Grids are typically organized into multiple administrative domains. Within each domain, one node is designated as information server to answer queries about all the resources belonging to that domain. Since information servers are highly reliable nodes, it is possible to build a P2P network of information servers having a significantly lower churn rate than typical P2P networks. Thus, we consider a scenario in which the DHT overlay is composed by information servers only, which ensures a high stability of the overlay even in large-scale networks.

The work that most relates to DQ-DHT is the Structella system designed by Castro et al. [10]. Structella replaces the random graph of Gnutella with the structured overlay of Pastry [11], while retaining the content placement of unstructured P2P systems to support complex queries. Queries in Structella are propagated using either constrained flooding or random walks. Each node receiving a query evaluates it against the local content and sends matching content back to the query originator. Beyond Structella, a few other works broadly relate to DQ-DHT for their combined use of structured and unstructured P2P techniques (see for example [12] and [13]).

In this paper we describe the DQ-DHT algorithm using Chord as DHT overlay. We analyze the performance of DQ-DHT through simulations under different algorithm configurations. We also compare the performance of DQ-DHT with that of DQ in unstructured networks. The simulation results show that DQ-DHT generates much less network overhead (i.e., number of messages) than DQ, with a comparable - and in some cases better - search time, and with a higher success rate when the resource to be found is rare.

The rest of the paper is organized as follows. Section 2 provides a background on the technique of broadcast over a DHT exploited by DQ-DHT. Section 3 describes the DQ-DHT algorithm. Section 4 analyzes its performance and compares DQ-DHT with DQ. Finally, Section 5 concludes the paper.

2 Broadcast over a DHT

This section briefly describes the Chord-based implementation of the broadcast algorithm designed by El-Ansary et al., as it is proposed in [8].

Chord uses a consistent hash function to assign each node an m-bit identifier, which represents its position in a circular identifier space ranging from 0 and $2^m - 1$. Each node, x, maintains a *finger table* with m entries. The j^{th} entry in the finger table at node x contains the identity of the first node, s, that succeeds x by at least 2^{j-1} positions on the identifier circle, where $1 \leq j \leq m$. Node s is called the j^{th} *finger* of node x. If the identifier space is not fully populated (i.e., the number of nodes, N, is lower than 2^m), the finger table contains redundant fingers. In a network of N nodes, the number u of unique (i.e., distinct) fingers of a generic node x is likely to be $\log_2 N$ [1]. In the following, we will use the notation F_i to indicate the i^{th} *unique finger* of node x, where $1 \leq i \leq u$.

To perform the broadcast of a data item D, a node x sends a BROADCAST message to all its unique fingers. The BROADCAST message contains D and a *limit* argument, which is used to restrict the forwarding space of a receiving node. The *limit* sent to F_i is set to F_{i+1}, for $1 \leq i \leq u-1$. The *limit* sent to the last unique finger, F_u, is set to the identifier of the sender, x. When a node y receives a BROADCAST message with a data item D and a given *limit*, it is responsible for forwarding D to all its unique fingers in the interval $]y$, *limit*$[$. When forwarding the message to F_i, for $1 \leq i \leq u-1$, y supplies it a new *limit*, which is set to F_{i+1} if it does not exceed the old *limit*, to the old *limit* otherwise. As before, the new *limit* sent to F_u is set to y.

As shown in [8], in a network of N nodes, a broadcast message originating at an arbitrary node reaches all other nodes after exactly $N-1$ messages, with $\log_2 N$ steps. The overall broadcast procedure can be viewed as the process of passing the data item through a spanning tree, rooted at the querying node, which covers all nodes in the network. Since the spanning tree corresponds to the lookup tree, which is a *binomial tree* in a (fully populated) Chord network [14], also the spanning tree associated to the broadcast over a fully populated Chord ring is a binomial tree.

3 Dynamic Querying over a DHT

In short, the DQ-DHT algorithm works as follows. Let x be the node that initiates the search, U the set of unique fingers not yet visited, and R_d the desired number of results. Initially U includes all unique fingers of x. Node x starts by choosing a subset V of U and sending the query to all fingers in V. These fingers will in turn forward the query to all nodes in the portions of the spanning tree they are responsible for, following the broadcast algorithm described above. When a node receives a query, it checks for local items matching the query criteria and, for each matching item, sends a query hit directly to x. The fingers in V are removed from U to indicate that they have been already visited.

After sending the query to all nodes in V, x waits for an amount of time T_L, which is the estimated time needed by the query to reach all nodes, up to a given level L, of the subtrees rooted at the unique fingers in V, plus the time needed to receive a query hit from those nodes. Then, if the current number of received query hits R_c is equal or greater than R_d, x terminates. Otherwise, an iterative procedure takes place.

At each iteration, node x: 1) calculates the item popularity P as the ratio between R_c and the number of nodes already theoretically queried; 2) calculates the number H_q of hosts in the network that should be queried to hit R_d query hits based on P; 3) chooses, among the nodes in U, a new subset V' of unique fingers whose associated subtrees contain at least H_q nodes; 4) sends the query to all nodes in V'; 5) waits for an amount of time needed to propagate the query to all nodes in the subtrees associated to V'.

The iterative procedure above is repeated until the desired number of query hits is reached, or there are no more fingers to contact. Note that, if the item popularity is properly estimated after the first phase of search, only one additional iteration may be sufficient to obtain the desired number of query hits.

An important point in DQ-DHT is estimating the number of nodes present in the different subtrees, and at different levels, of the spanning tree associated to the broadcast process. In the next section we discuss how we calculate such properties of the spanning tree and introduce some functions that are used in the algorithm (described in Section 3.2).

3.1 Properties of the Spanning Tree Associated to the Broadcast Process

As recalled in Section 2, the spanning tree associated to the broadcast over a fully populated Chord ring is a binomial tree. A binomial tree of order $i \geq 0$, B_i, consists of a root with i subtrees, where the j^{th} subtree is a binomial tree of order $j-1$, with $1 \leq j \leq i$. Given a binomial tree B_i, the following properties can be proven [15]: 1) The number of nodes in B_i is 2^i; 2) The depth of B_i is i; 3) The number of nodes at level l in B_i is given by the binomial coefficient $\binom{i}{l}$.

Given the binomial tree properties, we can calculate the properties of the spanning tree associated to a broadcast initiated by a node having u unique fingers (see Table 1).

Basically, in Table 1 we correct the binomial tree properties by a factor $c = N/2^u$, where N is the number of nodes in the network (which can be estimated [16]), to compensate the fact that the value of u may be different from the value of $\log_2 N$ in case of not fully populated rings. Note that, since the value of D_i may be not an integer, we use the generalized binomial coefficient to calculate N_i^l.

Table 1. Properties of the spanning tree rooted at a node with u unique fingers $F_1..F_u$

Notation	Description	Value
N_i	Number of nodes in the subtree rooted at F_i, where $1 \leq i \leq u$	$2^{i-1} \times c$
D_i	Depth of the subtree rooted at F_i, where $1 \leq i \leq u$	$\log_2 N_i$
N_i^l	Number of nodes at level l of the subtree rooted at F_i, where $1 \leq i \leq u$ and $0 \leq l \leq D_i$	$\binom{D_i}{l}$

Table 2. Aggregate functions operating on a set V of n unique fingers with indices $i_1..i_n \in [1, u]$

Function	Returned result	Value
$\mathrm{N}(V)$	Total number of nodes in the subtrees associated to the unique fingers in V	$\sum_{i=i_1..i_n} N_i$
$\mathrm{D}(V)$	Depth of the subtree associated to the unique finger with highest index in V	D_i where $i = \max(i_1..i_n)$
$\mathrm{N}(V, L)$	Total number of nodes from level 0 to level L of the subtrees associated to the unique fingers in V	$\sum_{i=i_1..i_n} \sum_{l=0}^{l_i} N_i^l$ where $l_i = \min(L, D_i)$

Based on the spanning tree properties defined in Table 1, we define in Table 2 some aggregate functions operating on a set of unique fingers. Such functions are used in the DQ-DHT algorithm presented in the next section.

3.2 DQ-DHT Algorithm

DQ-DHT defines two procedures: SubmitQuery, executed by a node to submit a query, and ProcessQuery, executed by a node receiving a query to process.

SubmitQuery (see Fig. 1) receives the query Q and the desired number of results R_d. It makes use of the functions defined in Table 2, and it is assumed that the procedure is executed by a node x.

The procedure starts by initializing to 0 the current number of results R_c (*line 1*). The value of R_c is incremented by 1 whenever a query hit is received. A set U is initialized to contain all unique fingers of node x (*line 2*), and H_t is set to $\mathrm{N}(U)$, which corresponds to the total number of hosts that can be queried in the network (*line 3*). The first subset V of fingers to visit is selected from U (*line 4*), and U is updated accordingly (*line 5*).

Afterwards, an integer L between 0 and $\mathrm{D}(V)$ is chosen (*line 6*). The value of L represents the last level of the subtrees associated to V from which to wait a response before to estimate the item popularity. The amount of time T_L needed to receive a response from those levels is then calculated as $T_H \times (L + 2)$, where T_H is the average time to pass a message from node to node (*line 7*). The value $L + 2$ is obtained by counting one hop to pass the message from x to the fingers, L hops to propagate the message up to level L, and an additional hop to return the query hit to node x.

```
procedure SubmitQuery($Q, R_d$)
 1: $R_c \Leftarrow 0$
 2: $U \Leftarrow$ all unique fingers of node $x$
 3: $H_t \Leftarrow \mathrm{N}(U)$
 4: $V \Leftarrow$ a subset of $U$
 5: $U \Leftarrow U \setminus V$
 6: $L \Leftarrow$ an integer $\in [0, \mathrm{D}(V)]$
 7: $T_L \Leftarrow T_H \times (L+2)$
 8: Send($Q, V$)
 9: sleep($T_L$)
10: $H_v \Leftarrow \mathrm{N}(V, L)$
11: $T_r \Leftarrow T_H \times (\mathrm{D}(V) - L)$
12: while $R_c < R_d$ and $U \neq \emptyset$ do
13:     if $R_c > 0$ then
14:         $P \Leftarrow R_c / H_v$
15:         $H_d \Leftarrow R_d / P$
16:     else
17:         $H_d \Leftarrow H_t + 1$
18:     end if
19:     if $H_d \leq \mathrm{N}(V)$ then
20:         sleep($T_r$)
21:         $H_v \Leftarrow \mathrm{N}(V)$
22:         $T_r \Leftarrow 0$
23:     else
24:         $H_q \Leftarrow H_d - \mathrm{N}(V)$
25:         if $H_q > \mathrm{N}(U)$ then
26:             $V' \Leftarrow U$
27:         else
28:             $V' \Leftarrow$ subset of $U$ with min. $\mathrm{N}(V') \geq H_q$
29:         end if
30:         $U \Leftarrow U \setminus V'$
31:         $T_{V'} \Leftarrow T_H \times (\mathrm{D}(V') + 2)$
32:         Send($Q, V'$)
33:         sleep($\max(T_{V'}, T_r)$)
34:         $H_v \Leftarrow \mathrm{N}(V) + \mathrm{N}(V')$
35:         $V \Leftarrow V'$
36:         $T_r \Leftarrow 0$
37:     end if
38: end while

subroutine Send($Q, V = \{F_{i_1}..F_{i_n}\}$)
 1: for $i = i_1$ to $i_n$ do
 2:     if $i < u$ then
 3:         limit $\Leftarrow F_{i+1}$
 4:     else
 5:         limit $\Leftarrow x$
 6:     end if
 7:     send message $M = \{x, Q, limit\}$ to node $F_i$
 8: end for
```

Fig. 1. The SubmitQuery procedure

Then, Q is sent to all fingers in V invoking the subroutine Send described below (*line 8*). After the wait (*line 9*), the number of nodes visited H_v is initialized to $\mathrm{N}(V, L)$ (*line 10*). While the popularity will be estimated considering only levels from 0 to L, the query continues to be forwarded up to level $\mathrm{D}(V)$. The additional amount of time T_r that would be necessary to get a response from the remaining levels is therefore proportional to $\mathrm{D}(V) - L$ (*line 11*).

After this first phase, an iterative process takes place while $R_c < R_d$ and there are more fingers to visit ($U \neq \emptyset$) (*line 12*). If at least one result has been received, node x estimates the item popularity P (*line 14*), and the estimated number H_d of hosts to obtain R_d results based on P (*line 15*). Otherwise (i.e., $R_c = 0$), H_d is set to $H_t + 1$, meaning that it is likely that *more than* all available hosts must be contacted to hit R_d results (*line 17*).

If $H_d < \mathrm{N}(V)$, it is expected to receive enough results from the fingers that have been already contacted. Note that this may happen only if $L < \mathrm{D}(V)$, because P is estimated on the basis of the results arriving from nodes up to level L of the subtrees associated to V. Thus, only in this case, the search initiator must wait for the additional amount of time T_r (*line 20*). After the wait, the value of H_v is updated to include all nodes in V (*line 21*), and T_r is set to 0 (*line 22*).

Otherwise ($H_d > \mathrm{N}(V)$), the number of nodes to be queried H_q is given by H_d minus the number of nodes already queried (*line 24*). If H_q is greater than the number of nodes available, the new set V' of fingers to visit is set to U (*line 26*).

```
procedure ProcessQuery($M = \{x, Q, limit\}$)
1:  for $i = 1$ to $u$ do
2:      if $F_i \in ]y, limit[$ then
3:          if $i < u$ then
4:              $oldLimit \Leftarrow limit$
5:              $limit \Leftarrow F_{i+1}$
6:              if $limit \notin ]y, oldLimit[$ then
7:                  $limit \Leftarrow oldLimit$
8:              end if
9:          else
10:             $limit \Leftarrow y$
11:         end if
12:         send message $M = \{x, Q, limit\}$ to node $F_i$
13:     else
14:         exit for
15:     end if
16: end for
17: for each local item matching $Q$ do
18:     send query hit to node $x$
19: end for
```

Fig. 2. The ProcessQuery procedure

Else, V' is the subset of U with the minimum value of $\mathrm{N}(V')$ which is greater or equal to H_q (*line 28*). The elements in V' are removed from U (*line 30*), and the time $T_{V'}$ needed to receive response from all levels of the subtrees associated to V' is calculated (*line 31*).

After sending the query to all nodes in V' (*line 32*), x performs a wait (*line 33*), updates the number of hosts visited (*line 34*), and sets V to V' (*line 35*). The waiting time on *line 33* is the maximum between $T_{V'}$ and T_r, for managing the case in which the time T_r needed to visit the levels remaining from the previous phase is greater than the time $T_{V'}$ needed to receive a response from all levels in V'. As for *lines 19-22*, this may happen only on the first iteration, since after that the timeout is always set to be proportional to $\mathrm{D}(V')$, and so $T_r = 0$ (*line 36*).

The subroutine Send forwards the query Q to a set of unique fingers V. Basically, it implements the procedure executed by a node x to perform a broadcast (see Section 2). The only difference is that we do not send the message to all unique fingers of x, but only to those in V. The message M sent by x to a node y includes the Id of the querying node (x), the query to be processed Q, and the *limit* parameter used to restrict the forwarding space of node y.

ProcessQuery (see Fig. 2) is executed by a node y that receives a message M containing the Id of the search initiator x, the query to process Q, and the *limit* parameter.

The procedure broadcasts the query to all nodes in the portion of the spanning tree node y is responsible for (*lines 1-16*), following the broadcast algorithm described in Section 2. Then, it processes the query against its local resources, and for each matching item sends a query hit directly to the search initiator (*lines 17-19*).

4 Performance Evaluation

We evaluate DQ-DHT in terms of two performance parameters: *number of messages* (N_m) and *search time* (T_s). N_m is the total number of messages generated during the search process, while T_s is the amount of time needed to receive the desired number of results.

The system parameters are: the number of nodes in the network (N) and the resource replication rate (r), where r is the ratio between the total number of resources satisfying the query criteria and N. The algorithm parameters are: the initial set of unique fingers to visit (V), the initial number of levels (L), and the desired number of results (R_d). Even if it is possible to choose V to include an arbitrary subset of the unique fingers of the querying node, we consider the case in which $V = \{F_i\}$, i.e., V includes only the i^{th} unique finger, where $1 \leq i \leq u$. This permits to have, after the probe query, still $u - 1$ unique fingers from which to choose the new set of subtrees to query, this way improving the granularity of search.

To analyze the message and time complexity of the algorithm we consider the following worst case scenario: at each iteration (including the probe query) the querying node chooses exactly one unique finger to contact, among those not yet contacted. Therefore, the overall search process will complete in u iterations. Since all subtrees are queried one after another, $N_m = N - 1$, and so the message complexity is $O(N)$. In the same scenario the search time is the sum of the times needed to query all subtrees in sequence, i.e., $T_s = T_H \times \sum_{i=1}^{u}(D_i + 2)$, where T_H is the average time per hop. From Table 1, $D_i = \log N_i = \log(2^{i-1} \times c) = i - 1 + \log c$, where $c = N/2^u$. Thus, $T_s = T_H \times \sum_{i=1}^{u}(i + 1 + \log c) = T_H \times (\frac{1}{2}u^2 + \frac{3}{2}u + (\log c)u)$. Since on average $u = \log N$, we obtain that $T_s = T_H \times (\frac{1}{2}\log^2 N + \frac{3}{2}\log N)$. Therefore, the time complexity in the worst case is $O(\log^2 N)$.

The worst case scenario considered above is based on the very pessimistic assumptions that, at each iteration, the current estimated value of the resource popularity determines the inclusion in the next set V of exactly one unique finger among those still available. In a more typical scenario, assuming a uniform distribution of the matching resources across nodes, the popularity can be estimated with enough accuracy during the probe query, thus allowing most searches to complete in two iterations. In such two-iteration scenario the maximum search time is the sum of the probe query time (which is proportional to $L + 2$) plus the time to cover the deepest subtree that can be chosen for the second iteration (i.e., the subtree associated to F_u): $T_s = T_H \times ((L + 2) + (D_u + 2))$. Since on average $D_u = \log N - 1$, $T_s = T_H \times ((L + 2) + (\log N + 1))$ and so the time complexity in such scenario is $O(\log N)$.

4.1 Simulation Analysis

We experimentally evaluated the behavior of DQ-DHT in different scenarios using a discrete-event simulator. All the tests have been performed in a randomly-generated Chord network with $N = 50000$ nodes and a value of r ranging from 0.25 % to 32 %. Different combinations of the algorithm parameters V, L, and R_d have been experimented. All the results presented in the following are calculated as an average of 100 independent simulation runs, where at each run the search is initiated by a randomly chosen node.

We run a first set of simulations to evaluate the behavior of DQ-DHT varying the initial set V of unique fingers to contact. At each run we chose V to include

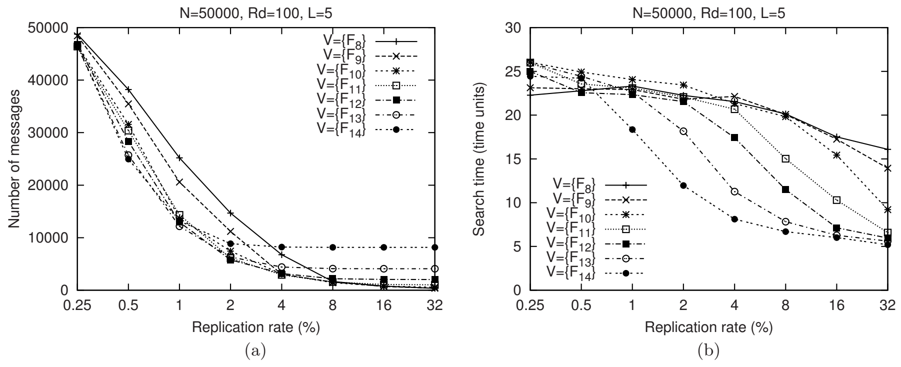


Fig. 3. Effect of varying the initial set V, with $L = 5$ and $R_d = 100$: (a) number of messages; (b) search time

one of the fingers between F_8 to F_{14}, with the initial value of L fixed to 5, and R_d set to 100. The graphs in Fig. 3 show number of messages and search time in function of the replication rate. The search time is expressed in time units, where one time unit corresponds to the average time to pass a message from node to node.

As expected, Fig. 3a shows that the number of messages decreases as the replication rate increases, for any value of V. When $V = \{F_8\}$, the average number of messages passes from 48735 for $r = 0.25\%$, to 360 for $r = 32\%$. In the opposite case, $V = \{F_{14}\}$, the number of messages passes from 46473 for $r = 0.25\%$, to 8159 for $r = 32\%$.

For high values of r (i.e., $r = 16 - 32\%$), in most cases the probe query is sufficient to obtain the desired number of results, and so the number of messages corresponds to the number of nodes in the subtree associated to the finger in V.

For values of r lower than 2%, typically at least one additional iteration after the probe query is needed. In these cases, the generated number of messages depends on the accuracy of the popularity estimation, which is better when a higher number of nodes is queried during the probe query (that is, when V includes a finger with a high index). For instance, when $r = 1\%$, the average number of messages is 25207 for $V = \{F_8\}$, 14341 for $V = \{F_{11}\}$, and 13169 for $V = \{F_{14}\}$.

This suggests to start the search by contacting a finger with a high index (e.g., F_{14}), when it is known that the resource is "rare." When there is no information about the popularity of the resource to be found, an intermediate finger (e.g., F_{11}) should be used.

As shown in Fig. 3b, also the search time decreases as the replication rate increases, for any value of V. When $V = \{F_8\}$, the average search time passes from 22.3 for $r = 0.25\%$, to 16.1 for $r = 32\%$. When $V = \{F_{14}\}$, the search time ranges from 24.4 for $r = 0.25\%$, to 5.2 for $r = 32\%$.

The graph shows that with low values of r it is convenient to contact a finger with a high index, which leads to a lower search time with respect to fingers with

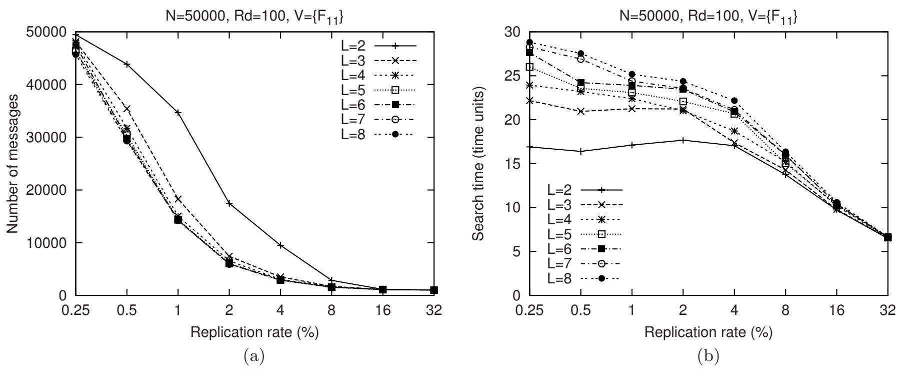


Fig. 4. Effect of varying the initial value of L, with $V = \{F_{11}\}$ and $R_d = 100$: (a) number of messages; (b) search time

a lower index. However, since the the main objective of DQ-DHT is reducing the number of messages, an intermediate finger (e.g., F_{11}) should be preferred in most cases, even if this may result to an increased search time.

We run a second set of simulations to evaluate the effect of varying the initial value of L. According to the results discussed above, we chose an intermediate finger for the probe query ($V = \{F_{11}\}$), and varied L from 2 to 8, with R_d fixed to 100. The results are presented by the graphs in Fig. 4.

Fig. 4b shows that lower values of L generate lower search times. For instance, when $r = 1\%$ the average search time passes from 17.1 with $L = 2$, to 25.2 with $L = 8$. This is mainly due to the fact that the wait after the probe phase is proportional to L, as described in Section 3.2.

On the other hand, Fig. 4a shows that very low values of L produce a significant increase in the number of messages. For example, when $r = 1\%$ the average number of messages passes from 14259 with $L = 8$, to 34654 with $L = 2$. The excess of messages in the second case is due to the reduced accuracy in the estimation of the resource popularity that is obtained considering only a few levels of the subtrees associated to V.

In general, intermediate values of L produce the best compromise between number of messages and search time. For the scenario analyzed here ($V = \{F_{11}\}$), the best result is obtained with $L = 4$, which generates a number of messages similar to that produced by higher values of L, but with a quite lower search time, as shown by the graphs in Fig. 4.

4.2 Comparison with Dynamic Querying in Unstructured Networks

In this section we compare the performance of DQ-DHT with that of DQ in unstructured networks. Since DQ-DHT is designed to work on a DHT-based network, while DQ works on unstructured networks, we adopted the following approach to compare the two systems. First, we built a random Chord network with $N = 50000$ nodes, and measured the average number of unique fingers across all nodes, which resulted to be $\bar{u} = 15.94$. Then, we built an unstructured overlay among the same nodes, in which each node is connected to $\bar{u}$ other random nodes, on the average.

As before, we measured the *number of messages* and the *search time*. In addition, we evaluated the following performance parameters: *duplication rate*, defined as the percentage of duplicate messages on the total number of messages; *success rate*, defined as the percentage of successful searches on the total number of searches performed.

For DQ in unstructured networks, we implemented the DQ+ algorithm proposed by Jiang and Jin in [17], which is an enhanced version of the original algorithm proposed by Fisk in [9]. The main difference between DQ+ and the original DQ algorithm is briefly described in the following.

In the original DQ, after each iteration, the querying node calculates the total number H_t of hosts to query to reach the desired number of results. Then, it calculates the number H_n of hosts to query per neighbor as H_t/n, where n is the number of neighbors that have not yet received the query. Finally, it calculates the minimum TTL to reach H_n hosts through the next neighbor, and sends the query towards that neighbor.

DQ+ adopts a "greedy" strategy. After each iteration, the querying node estimates the total number H_t of host to query, and then calculates the minimum TTL to reach H_t hosts via the next neighbor alone. To avoid overshooting of the search space, DQ+ uses a confidence interval method to estimate the popularity of the searched item. The simulation results presented in [17] show that DQ+ reduces the latency by more than four times with respect to the original DQ algorithm.

The initial parameters of DQ+ are: the number of neighbors contacted during the probe phase, n, and the TTL used for the probe query, t. We experimented two configurations: *i)* $n = 3$ and $t = 2$; *ii)* $n = 3$ and $t = 3$. In both cases, the maximum value of TTL allowed after the probe query is 5.

For DQ-DHT we chose the following configurations: *i)* $V = \{F_{14}\}$ and $L = 5$; *ii)* $V = \{F_{11}\}$ and $L = 4$. The first configuration aims at minimizing the search time, but at the cost of a higher number of messages. The second configuration provides a better balance between number of messages and search time, as discussed above.

The results of the comparison between DQ-DHT and DQ+ are presented in Fig. 5. All the simulations have been conducted with a value of r ranging from 0.25 % to 32 %, and R_d fixed to 100. Moreover, each result is obtained as the average of 100 independent simulation runs.

As shown in Fig. 5c, the success rate for replication rates greater or equal to 0.5% is 100% with both DQ+ and DQ-DHT. However, for $r = 0.25\%$, DQ-DHT has a success rate of 100%, while DQ+ has a success rate of only 8.5%. This is due to the incomplete network coverage of the constrained flooding implemented by DQ+, which in some cases fails to find the desired number of results even if they are actually available in the network. On the contrary, DQ-DHT ensures a

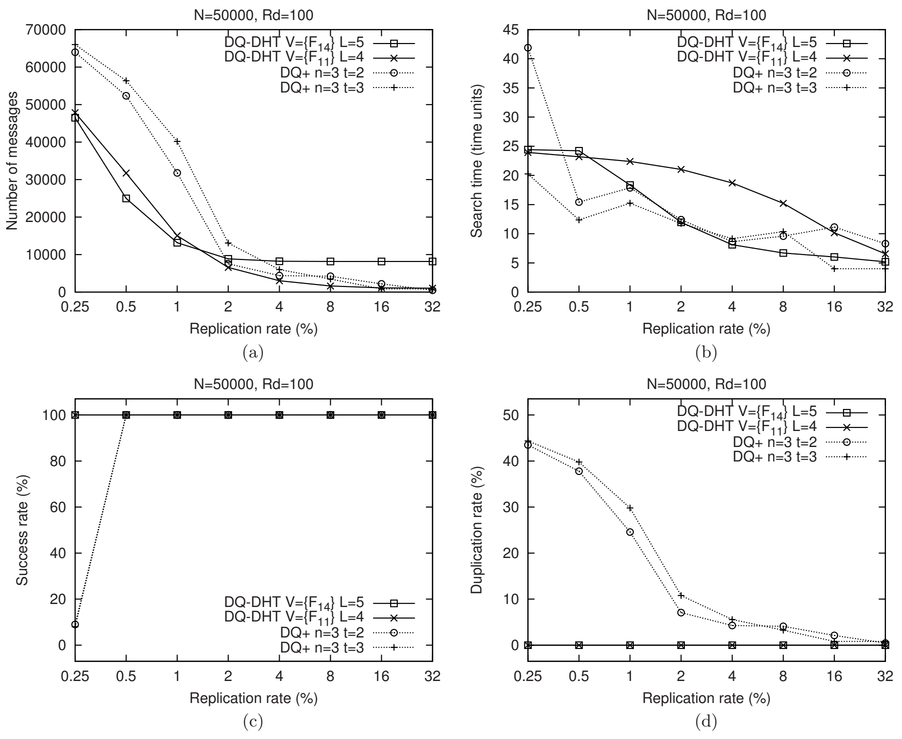


Fig. 5. Comparison between DQ-DHT and dynamic querying in unstructured networks (DQ+): (a) number of messages; (b) search time; (c) success rate; (d) duplication rate

complete network coverage and therefore maintains a success rate of 100% even in presence of very low replication rates.

The search time in DQ+ and DQ-DHT is compared in Fig. 5b. As already discussed above, the search time in DQ-DHT strongly depends on the choice of the initial set V. With $V = \{F_{14}\}$ the search time of DQ-DHT is comparable with (and in some cases better than) the search time of DQ+. This is obtained at the cost of more messages than the case in which $V = \{F_{11}\}$, but they are much less than those generated by DQ+ for values of $r < 2\%$.

Fig. 5a shows the number of messages generated by the two algorithms. DQ-DHT with $V = \{F_{11}\}$ produces less messages than DQ+ for all values of $r \leq 16\%$, while they generate approximatively the same number of messages for $r = 32\%$. For values of r lesser than 2% DQ-DHT outperforms DQ+ by more than a factor two. For instance, for $r = 1\%$ DQ-DHT with $V = \{F_{11}\}$ generates 15025 messages, while DQ+ with $t = 3$ produces 40155 messages.

Note that, for low replication rates, DQ-DHT generates less messages with $V = \{F_{14}\}$, while it works better with $V = \{F_{11}\}$ for high replication rates. This is due to the fact that with $V = \{F_{14}\}$ the minimum number of messages sent to the network is higher, and so more messages than needed are generated when the resource to be found is popular. On the other hand, a high number of messages ensures a better accuracy in the estimation of the resource popularity, leading to less messages when the resource to be found is rare.

The greater number of messages generated by DQ+ with respect to DQ-DHT is mainly due to the message duplication caused by flooding. The percentage of duplicate messages on the total number of messages is shown in Fig. 5d. As expected, the duplication rate of DQ+ increases as the replication rate decreases, reaching approximatively the value of 44% with $r = 0.25\%$. DQ-DHT does not suffer the message duplication problem, as each node receives the query at most once. Therefore, the duplication rate for DQ-DHT is 0% for any value of r.

In summary, the simulation results presented throughout this section show that DQ-DHT produces much less network overhead (i.e., number of messages) than DQ+, with a comparable - and in some cases better - search time, and with a higher success rate when the resource to be found is rare.

5 Conclusions

A way to support arbitrary queries in structured networks is implementing unstructured search techniques on top of DHT-based overlays. Following this approach, we proposed DQ-DHT: a P2P search algorithm that combines the dynamic querying technique with an algorithm for efficient broadcast over a DHT. DQ-DHT has been particularly designed to be used in Grid scenarios, where it is necessary to support arbitraries queries for searching resources on the basis of complex criteria or semantic features.

The behavior of DQ-DHT has been analyzed through a simulator by varying its initial configuration, in order to understand which are the best parameters to use based on user/system requirements and objectives (i.e., minimizing the number of messages or the search time). We also compared the performance of DQ-DHT with that of the enhanced dynamic querying in unstructured networks (DQ+). The simulation results show that DQ-DHT generates much less network overhead than DQ+, with a comparable (and in some cases better) search time, and with a higher success rate when the resource to be found is rare.

Acknowledgements

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

References

1. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM SIGCOMM 2001, San Diego, USA (2001)
2. Gnutella Protocol Development. `http://rfc-gnutella.sourceforge.net`

3. Andrzejak, A., Xu, Z.: Scalable, Efficient Range Queries for Grid Information Services. In: 2nd IEEE Int. Conf. on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden (2002)
4. Cai, M., Frank, M.R., Chen, J., Szekely, P.A.: MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Journal of Grid Computing 2(1), 3–14 (2004)
5. Harren, M., Hellerstein, J.M., Huebsch, R., Loo, B.T., Shenker, S., Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 242–250. Springer, Heidelberg (2002)
6. Castro, M., Costa, M., Rowstron, A.: Debunking Some Myths About Structured and Unstructured Overlays. In: 2nd Symp. on Networked Systems Design and Implementation (NSDI 2005), Boston, USA (2005)
7. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making Gnutella-like P2P Systems Scalable. In: ACM SIGCOMM 2003, Karlsruhe, Germany (2003)
8. El-Ansary, S., Alima, L., Brand, P., Haridi, S.: Efficient Broadcast in Structured P2P Networks. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 304–314. Springer, Heidelberg (2003)
9. Fisk, A.: Gnutella Dynamic Query Protocol v0.1 (2003), http://www9.limewire.com/developer/dynamic_query.html
10. Castro, M., Costa, M., Rowstron, A.: Should we build Gnutella on a structured overlay? Computer Communication Review 34(1), 131–136 (2004)
11. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218. Springer, Heidelberg (2001)
12. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The Case for a Hybrid P2P Search Infrastructure. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279. Springer, Heidelberg (2005)
13. Zaharia, M., Keshav, S.: Gossip-based Search Selection in Hybrid Peer-to-Peer Networks. In: 5th Int. Workshop on Peer-to-Peer Systems (IPTPS 2006), Santa Barbara, USA (2006)
14. Chou, J.C.Y., Huang, T.-Y., Huang, K.-L., Chen, T.-Y.: SCALLOP: A Scalable and Load-Balanced Peer-to-Peer Lookup Protocol. IEEE Trans. Parallel Distrib. Syst. 17(5), 419–433 (2006)
15. Preiss, B.R.: Data Structures and Algorithms with Object-Oriented Design Patterns in C++. John Wiley & Sons, Chichester (1998)
16. Binzenhöfer, A., Staehle, D., Henjes, R.: Estimating the size of a Chord ring. Technical Report 348, Institute of Computer Science, University of Würzburg (2005)
17. Jiang, H., Jin, S.: Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks. In: 13th IEEE Int. Conf. on Network Protocols (ICNP 2005), Boston, USA (2005)

Web-Based Management of Content Delivery Networks

George Oikonomou and Theodore Apostolopoulos

Athens University of Economics and Business,
Department of Informatics,
Athens, Greece
{geo,tca}@aueb.gr

Abstract. Abstract overlay networks have been considered enablers of efficient management for decentralized, large scale service deployments. A Content Delivery Network (CDN) is an example of service falling within this category. The result of our research is WebDMF, a management framework for distributed services based on the Web-Based Enterprise Management (WBEM) family of specifications. Abstract design, combined with a middleware layer of entities called "Representatives", makes WebDMF suitable for the management of a variety of services. Details related to the management of each particular service are detached from the representative logic. This paper discusses how WebDMF can be used for the management of CDNs. A WBEM provider resides on each host participating in the service deployment and implements CDN-specific operations. WebDMF representatives decentralize, unify and coordinate those on a deployment scale. Preliminary measurements on an emulated network topology are also presented as an indication of the solution's viability and scalability.

Keywords: Content Delivery Networks, Web-Based Enterprise Management, WebDMF, Distributed Services Management.

1 Introduction

The current paradigm of large scale decentralized service deployments poses new requirements in the field of network and systems management. Legacy approaches, such as the Simple Network Management Protocol (SNMP) [1] focus on single nodes and are best suited for devices. However, SNMP is considered less appropriate for applications and services. Nowadays, abstract overlay networks are considered enablers of distributed services management. Techniques used to access resources should be suitable for applications as well as devices.

WebDMF, the result of our research, is a Web-based Management Framework for Distributed Services. Its core is based on the Web-based Enterprise Management (WBEM) family of specifications [2,3,4]. It is not limited to monitoring but is also capable of modifying the run-time parameters of the managed service. Due to its abstract design it has wide scope and is suitable for the management of a variety of services.

WebDMF's development consisted in the design of a CIM extension schema and the implementation of a WBEM provider enabling CIM operations on classes of this schema. The framework's design, implementation and performance evaluation have been discussed in detail in [5]. We have particularly studied its application on the management of grids and web server load-balancing schemes.

This paper extends our previous work by discussing how WebDMF can be used for the management of Content Delivery Networks (CDNs). An open source tool (OpenCDN) was used to install a test CDN deployment and perform measurements. In this context, this paper's main contribution is two-fold:

– We designed an extension CIM schema for OpenCDN hosts. For this schema, we implemented a WBEM provider offering management facilities for hosts participating in a CDN service deployment.
– We demonstrate how we can achieve management of the entire CDN deployment by combining WebDMF operations with the aforementioned provider.

Section 2 briefly outlines existing efforts in the field of management of content delivery networks. In order to familiarize the reader with some basic concepts, the section continues with an introduction to the WBEM standards family and the OpenCDN tool. In Sect. 3 we present an overview of WebDMF's architectural design. In Sect. 4 we disclose the design and implementation of a WBEM provider for OpenCDN hosts. Furthermore we demonstrate how this provider can be integrated with a WebDMF network in order to perform management of the OpenCDN deployment. Section 5 presents an experiment indicative of the framework's scalability and in Sect. 6 we discuss our conclusions.

2 Related Work

In the field of Content Delivery (or Distribution) Networks, the term 'management' is often used to describe a variety of different concepts. A theoretic approach focusing on *content consistency* management in caching schemes has been discussed in [6]. In [7], the authors present a "*policy-based* architecture for the control and management of content distribution networks". An agent-based distributed management framework, using SNMP for data gathering from nodes has been presented in [8]. Finally, UPGRADE-CDN is a platform for the development of CDNs. It introduces a *monitoring* framework called "AMonitor" [9]. GLOBULE [10] is a collaborative content delivery network. This effort provides approaches and suggests solutions to large-scale, CDN-related problems. Among other topics, the authors discuss *content replication and consistency* management.

WebDMF goes past simple monitoring by providing active *configuration* management capabilities. Furthermore, in the context of this paper, the managed resource is the deployment itself. This includes content, network links as well as hosts participating in the deployment.

2.1 Web-Based Enterprise Management

Web-Based Enterprise Management (WBEM) is a set of management specifications published by the Distributed Management Task Force (DMTF). WBEM is made up of three core components.

1. Model: The "Common Information Model" (CIM) is an object-oriented, platform-independent approach for the modeling of management data [2]. It includes a "core schema" with definitions that apply to all management areas. It also includes a set of "common models" that represent common management areas, such as networks, hardware, software and services. Finally, the CIM allows manufacturers to define technology-specific "extension schemas" that directly suit the management needs of their implementations.
2. Encoding: WBEM adopts the client-server paradigm. For the interaction between entities (clients and managed elements), it uses a set of well-defined request and response data packets. CIM elements are encoded in XML in accordance with the xmlCIM specification [3].
3. Transport: The resulting XML document is transmitted over the network as the payload of an HTTP message. This transport mechanism is called "CIM Operations over HTTP" [4].

The term CIM-XML is often used to refer to the combination of 2 and 3 above.

A WBEM server is made up of components as portrayed in Fig. 1. The WBEM client does not have direct access to the managed resources. Instead, it sends requests to the CIM Object Manager (CIMOM), using CIM over HTTP. The CIMOM handles all communication with the client. It delegates requests to the appropriate providers and returns responses. Providers act as plugins for the CIMOM. They are responsible for the realization of management operations for a resource. Therefore, providers are implementation-specific. The repository is the part of the WBEM server that stores definitions of the core, common and extension CIM schemas.

A significant number of vendors have started releasing WBEM products. The SBLIM open source project offers a suite of WBEM-related tools. Furthermore, OpenPegasus, OpenWBEM and WBEMServices are examples of open source CIMOM implementations. There are also numerous commercial solutions.

2.2 Open Source Content Delivery Network Software

The "Open Content Delivery Network" (OpenCDN) project is an open source effort aiming to provide a platform for the deployment of content delivery services for streaming multimedia. It operates by creating an overlay network of hosts. These hosts stream content using an application layer multicast scheme. An OpenCDN deployment is made up of three components:

– Origins: Nodes that host the original multimedia content.
– Request Routing and Distribution Manager (RRDM): A centralized control entity that coordinates the streaming of content between nodes.

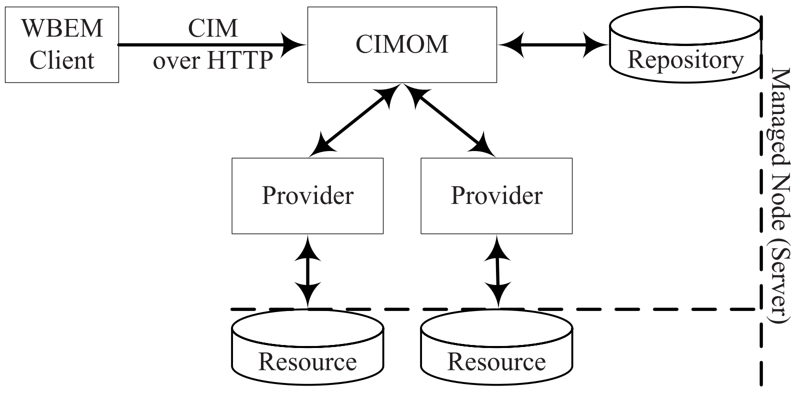


Fig. 1. WBEM instrumentation

– Distribution Nodes (or Nodes): These are the hosts that perform stream relay from the Origin to the final recipients.

Distribution nodes are organized hierarchically. Depending on their location in the hierarchy they are classified as "First Hop", "Transit" or "Last Hop". OpenCDN is written in PERL and uses XML-RPC messages for the communication between distribution nodes, origins and the RRDM. Adapters with existing streaming technologies are used in order to achieve content relay. New adapters can be written in order to provide support for more technologies. Further information and technical details on OpenCDN's design can be found in [11].

3 WebDMF: A Framework for the Management of Distributed Services

WebDMF stands for Web-based Distributed Management Framework. It treats a distributed deployment as a number of host nodes, called "Service Nodes". They are interconnected over a network and share resources to provide services to the end user. This section summarizes the framework's architecture and is meant as an overview. Detailed technical design has been disclosed in [5].

WebDMF's design is based on the WBEM family of technologies. Nodes function as WBEM entities; clients, servers or both, depending on their role in the deployment. Communication between nodes is performed in accordance with CIM-XML (see Sec. 2.1). WebDMF introduces a middleware layer of entities called "Representatives", forming a management overlay network. Representatives issue WBEM requests to perform management operations on service nodes. Once responses have been collected, they are unified in order to provide an abstract view of the deployment to the user. This resembles the "Manager of Managers" (MoM) approach. However, in MoM there is no direct communication between domain managers. In WebDMF, representatives are aware of their peers and communicate with them. Therefore WebDMF falls into the “Distributed Management” category.

3.1 WebDMF Entities

A “Management Node” corresponds to a typical WBEM client. It is used to monitor and configure various operational parameters of the distributed service. Any existing WBEM client software can be used without modifications.

A “Service Node” is the term used when referring to any node - member of the distributed service. For instance, in the case of a content delivery network a service node is an intermediate relay or a node hosting content. A node’s role in a particular distributed deployment does not affect WebDMF’s functionality. A service node executes an instance of the (distributed) managed service and a CIM Object Manager. WebDMF requests traverse the overlay network and eventually reach CIMOMs on service nodes. This is where management operations take place.

WebDMF introduces an entity called the “Management Representative”. This entity receives requests from a WBEM client (management node) and performs management actions on the relevant service nodes. After a series of message exchanges, it will respond to the initial request. A representative is more than a simple ‘proxy’ that receives and forwards requests. It is further split into building blocks, as shown in Fig. 2. It can act as a WBEM server as well as a client. Initial requests are received by the CIMOM on the representative. They are delegated to the WebDMF provider module for further processing. The module performs the following functions:

– Determines whether the request can be immediately served.
– If the node can not directly serve the request then it selects an appropriate representative and forwards it.

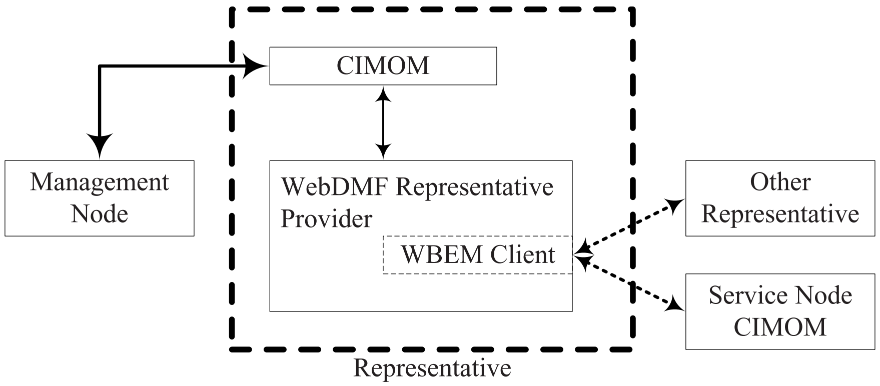


Fig. 2. The WebDMF representative. The *solid line* corresponds to the initial request and the final response whereas *dashed lines* represent intermediate request-response exchanges.

– If the request can be immediately served, the representative creates a list of service nodes that should be contacted and issues intermediate requests.
– Intermediate responses are processed and a final response is generated.
– Finally, a representative maintains information about the distributed service's topology.

In a WebDMF deployment, a representative is responsible for the management of a group of service nodes. The term "Domain" is used when referring to such groups. Domains are organized in a hierarchical structure. The hierarchy's root node corresponds to the entire deployment. The rationale behind designing the domain hierarchy of each individual deployment can be based on a variety of criteria. For example a system might be separated into domains based on nodes geographical locations. More sophisticated clustering techniques can also be used.

3.2 WebDMF Operations and CIM Schemas

WebDMF defines two categories of management operations: i) horizontal and ii) vertical. Horizontal operations enable management of the WebDMF overlay network itself. Those functions can, for example, be used to perform topology changes. The message exchange taking place does not involve service nodes and the managed service is not affected in any way. On the other hand, vertical operations read and modify CIM instances on service nodes, thus achieving management of the target application. Some common examples include:

– Setting new values on CIM objects of many service nodes.
– Reading operational parameters from service nodes and reporting an aggregate (e.g. sum or average).

In line with the above, we have designed two CIM Schemas for WebDMF, the core schema (`WebDMF_Core`) and the request factory. They both reside on the representatives repositories. The former schema models the deployment's logical topology, as discussed earlier. It corresponds to horizontal functions.

The latter schema corresponds to vertical functions. Users can call WBEM methods on instances of this schema. In doing so, they can define management operations that they wish to perform on the target application. Each request towards the distributed deployment is treated as a managed resource itself. For example, users can create a new request. They can execute it periodically and read the results. They can modify it, re-execute it and finally delete it. Each request is mapped by the representative to intermediate WBEM requests issued to service nodes. Request factory classes are generic. They are not related in any way with the CIM schema of the managed application. This makes WebDMF appropriate for the management of a wide variety of services. Furthermore, the request factory does not need re-configuration when the target schema is modified.

4 Management of Content Delivery Networks

4.1 A WBEM Provider for OpenCDN Hosts

In order to achieve WBEM management of an OpenCDN host, two things are required: i) a CIM schema modeling management information and ii) a WBEM provider. That provider should be OpenCDN-specific in order to be capable of implementing operations on the managed resource. In this case, the managed resource is a process running on the host, offering the service.

As part of our work, we designed a CIM schema for OpenCDN hosts. As displayed in Fig. 3, the schema is made up of 9 classes. Seven of the classes correspond to hosts, one class represents multimedia content and finally one class represents a CIM association. Only class names and relationships are displayed in the figure. Properties and Methods are omitted for clarity reasons. Class `oCDN_Host` defines some properties and methods common to all OpenCDN host types. Similarly, class `oCDN_Node` defines attributes common to distribution nodes, regardless of relay technology. These two classes are abstract (italicized names) therefore can not have instances. Each instance of a non-abstract class corresponds to a different host (or program).

Following the design of a CIM schema for OpenCDN, we implemented a provider for classes `oCDN_DNode`, `oCDN_Origin` and `oCDN_RRDM`. By calling WBEM intrinsic methods on their instances, the user can achieve management of the respective type of host. For example, the user can restart the DNode service or

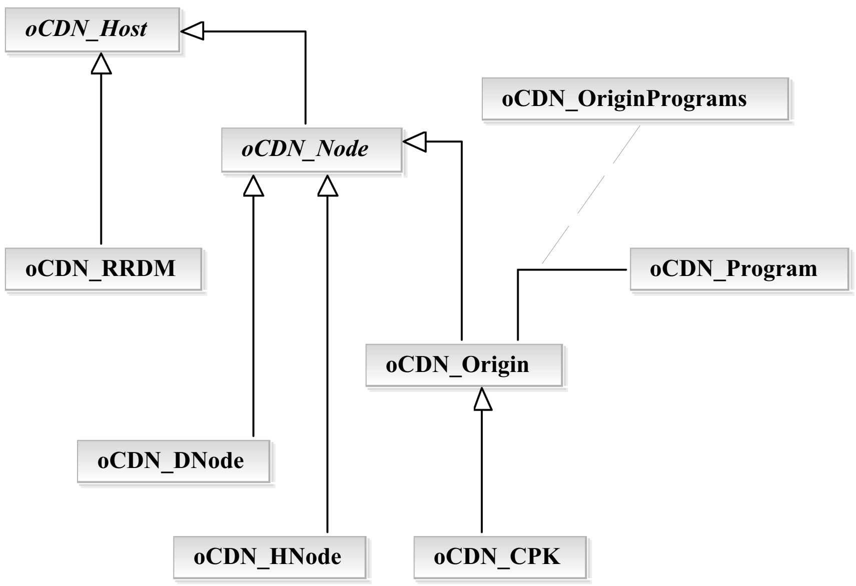


Fig. 3. CIM schema for the management of OpenCDN hosts

modify its runtime parameters. To be more specific, the provider implements the following functions:

– Start - stop service. This is performed by calling the `CreateInstance` and `DeleteInstance` intrinsic WBEM methods respectively.
– Retrieve the names of all service instances running on a particular host. This is achieved by the `EnumerateInstanceNames` method. Only instance identifiers are returned by this method, without detailed property values.
– Read the configuration of a running service. The `GetInstance` method is used to achieve this. Furthermore, if more than one service is offered by a host (e.g. RRDM and DNode concurrently) then calling the `EnumerateInstances` method results in a response containing the parameter values for all services. Each service is represented by a separate CIM instance.
– Modify the configuration of a particular instance. This happens by calling the `ModifyInstance` WBEM method.

The figure below (Fig. 4) displays a UML activity diagram for the case of configuration modification. The user issues a `ModifyInstance` request which is received by the CIMOM. If no errors occur during processing, the following steps take place:

1. The CIMOM invokes the OpenCDN provider and delegates the request.
2. The provider checks the requested changes' validity.
3. The provider maintains information of all running services in an instance pool. Each instance corresponds to a different service. At this step the provider searches the pool for the requested instance.
4. The provider performs changes on the instance. It also modifies the configuration of the running process that offers the service and notifies the CIMOM.
5. A response is generated by the CIMOM and sent to the management station, indicating successful completion of the management operation.

4.2 Integrating an Open CDN Deployment with WebDMF

The WBEM provider discussed in the previous section handles all management requests for a single OpenCDN host. Its design is directly related to OpenCDN's implementation details and it can perform management operations directly on resources. However, an OpenCDN deployment is made up of multiple hosts. This is usually one RRDM and multiple origin and distribution nodes. Even with the OpenCDN provider installed on each of them, they would still be treated as stand alone entities in a typical WBEM-based management infrastructure. WebDMF adds an abstraction layer that unifies the hosts as parts of a single service.

In WebDMF terminology (see Sect. 3), all OpenCDN nodes are treated as "service nodes". In order to integrate the CDN deployment with a WebDMF management overlay, two steps need to be performed:

– The OpenCDN network needs to be broken down into domains. This is a change in logical terms but does not cause any alterations on service nodes.

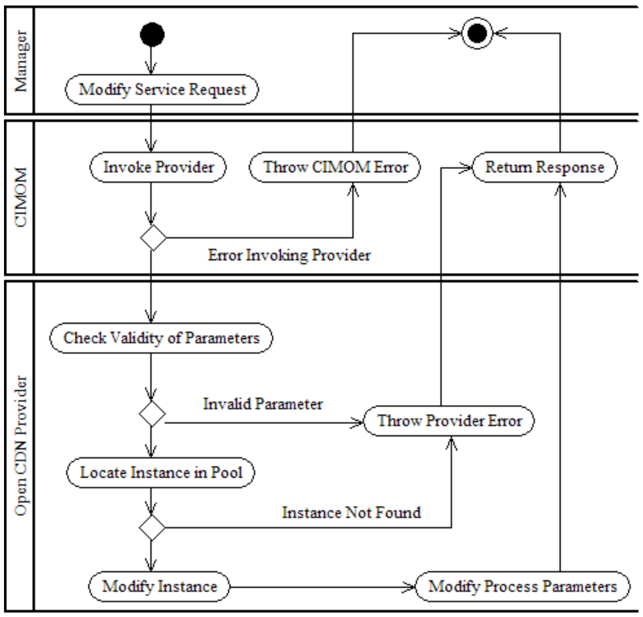


Fig. 4. UML activity diagram modeling a `ModifyInstance` operation. This operation involves three actors (the CIMOM, the OpenCDN provider and the management station). *Swimlanes* are used to group activities performed by the same actor.

– Representatives need to be installed and assigned for the aforementioned domains. Nodes are registered on the representatives through WebDMF *horizontal* operations, thus generating a virtual topology for the deployment.

Figure 5 displays a sample OpenCDN topology. Shapes indicate a node's role in the deployment (RRDM, relay or origin). Arrows show the path that a multimedia stream would follow from the origin node towards the final viewer. This topology has been broken down into three domains, different fill textures are used to distinguish those. This domain structure adopts a two level hierarchy. The top level (root of the tree) corresponds to the entire deployment. The second level is the one displayed in the figure. This domain structure is just indicative. It would be possible to use a multi-level hierarchy with super-domains and sub-domains. For example Domains 1 and 2 could both be children of a single parent. In this example topology, 3 representatives are being used, displayed with hexagons and not linked with arrows to any other nodes. In line with the above, different fill textures indicate which domains they have been assigned to.

Once the above two steps have been completed, a user can issue management requests for the CDN by using WebDMF *vertical* facilities, as outlined in Sec. 3.2. Representatives communicate among themselves registering changes in the CDN deployment. They also map WebDMF requests to WBEM operations for classes of the OpenCDN CIM schema on the appropriate service nodes

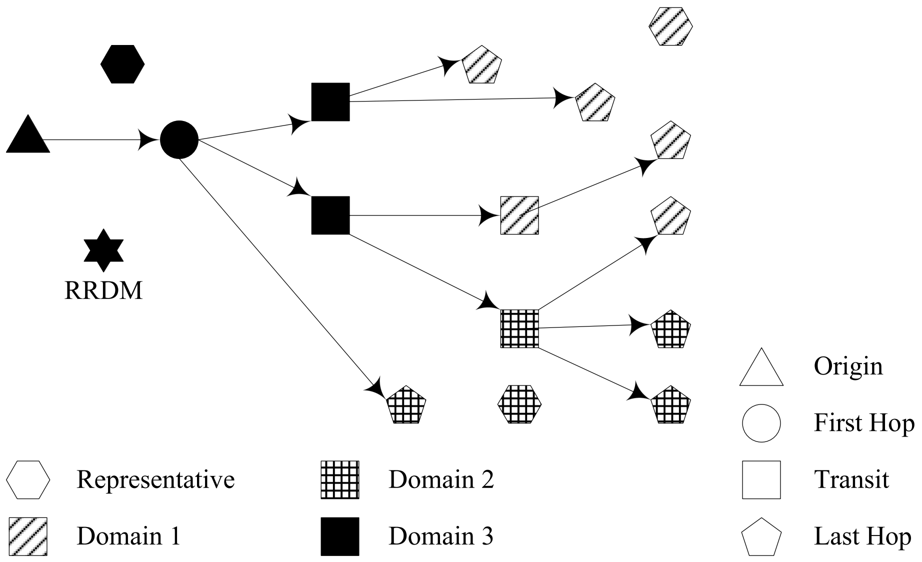


Fig. 5. A sample OpenCDN deployment broken down into three WebDMF domains

(Sec. 4.1). Changes on nodes are performed by the OpenCDN WBEM provider that resides on each node's CIMOM.

4.3 Implementation Details

The WebDMF representative is implemented as a single shared object library file (.so). It is comprised of a set of WBEM providers, each one of them implementing management operations for a class of the WebDMF schemas. The OpenCDN provider is also a single shared object file implementing WBEM operations for classes `oCDN_Origin`, `oCDN_RRDM` and `oCDN_DNode`. The interface between the CIMOM and providers complies with the Common Manageability Programming Interface (CMPI) [17]. Providers themselves are written in C++. This does not break CIMOM independence, as described in [17]. The representative was developed on Linux 2.6.20 machines. We used gcc 4.1.2 and version 2.17.50 of binutils. Testing took place using version 2.7.0 of the Open Pegasus CIMOM.

5 Performance Evaluation

In order to evaluate WebDMF, we installed a testbed environment using ModelNet [12] and executed various measurements. Results were obtained from actual code execution on an emulated network topology, they are not simulation results. In this section we present the outcome of an experiment that is indicative of WebDMF's scalability.

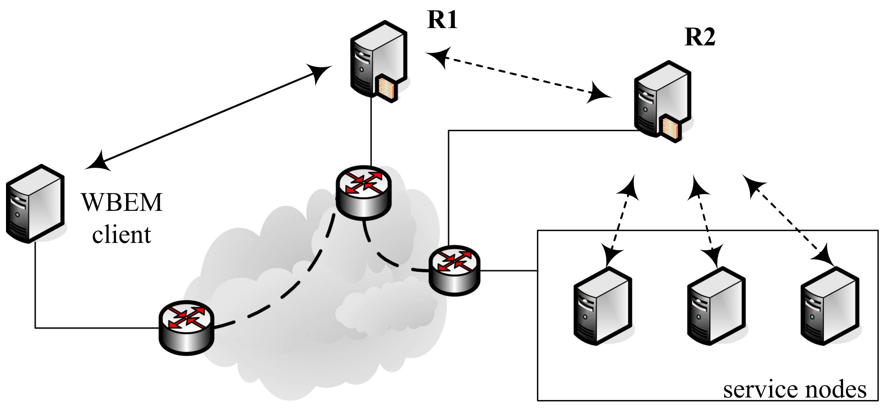


Fig. 6. Emulated topology and test scenario

The topology emulated by ModelNet represents a wide-area network. It consists of 300 virtual nodes situated in 3 LANs with each LAN having its own gateway to the WAN. The 3 gateways are interconnected via a backbone network, with high bandwidth, low delay links. We also installed two WebDMF representatives (nodes R1 and R2). In our experiment, service nodes are OpenCDN origin nodes. The management operation constitutes of setting them to register with a different RRDM. This involves changing property values of the `oCDN_Origin` instances on service nodes. A client issues an initial WebDMF request to representative R1 (this is a WBEM `CreateInstance` for class `WebDMF_RequestWBEM` of the request factory). R1 forwards the request to R2. R2 sends `ModifyInstance` requests to service nodes. Responses follow the reverse path. Figure 6 portrays the emulated topology and test scenario. For clarity reasons, we omit service nodes residing in other domains.

In this experiment we wish to evaluate the impact of service node count on the time needed to complete the management operation described above. We repeated the experiment by gradually increasing the number of service nodes involved in the operation. Starting with 10 service nodes, we increased the number by 10 up to 270 (27 steps). For each step we executed the management operation 100 times. For each repetition we measured elapsed time with microsecond accuracy. This resulted in a total sample of 2700 observations (27 steps x 100 repetitions). Figure 7 displays a scatter-plot of the observed values. The X axis displays the number of service nodes involved in the operation. The Y axis corresponds to the total time needed to complete the management operation.

Based on the same sample we calculated a simple linear regression line using the number of nodes as the independent variable. The resulting regression line is plotted on the same diagram. For this regression, the R^2 coefficient equals 0.993, indicating strongly linear co-relation. Furthermore, residuals are distributed normally, indicating good fit of the regression line on the sample. More analysis of the same sample and goodness-of-fit tests for this regression line on a second

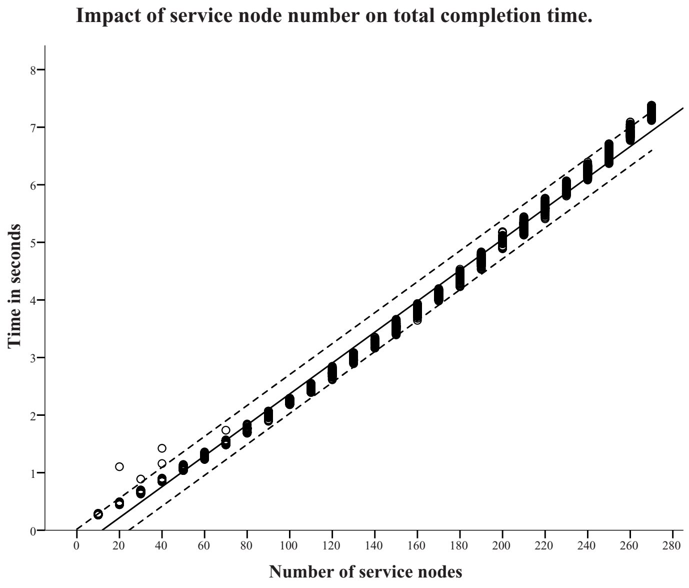


Fig. 7. Total completion time by number of service nodes. *Circles* represent observed values. The *solid line* represents a linear regression whereas the *dashed lines* display confidence intervals for the estimated values. Most of the observed values fall within the boundaries.

sample verify the validity of those findings. Those results are not disclosed here due to space limitations.

This experiment shows that under similar network load conditions, the time needed to complete a WebDMF vertical operation is linearly related to the number of service nodes that a representative will have to contact. This is an indication of the framework's scalability in dense service deployments.

6 Conclusions

In this paper we demonstrated how WebDFM can be applied in the field of Content Delivery Network management. This case study can be used to bring out some of the framework's advantages. Its open standards-based design facilitates its integration with existing WBEM infrastructures. WBEM relies on web technologies (HTTP for transport and XML for content encoding). This provides alternatives for security related problems such as firewall traversal. Existing encryption techniques, such as HTTP over SSL, can be used off the shelf, without need for modifications on the framework itself. WebDMF's design is abstract. By detaching the details of the managed service from the representative logic, it is generic and suitable for the management of a wide variety of services.

WebDMF is resource-centric, something that may seem to be a step in the opposite direction compared to emerging, web service-based efforts [13,14].

However, those initiatives are model-agnostic. They do not define properties and operations for the managed resources [14]. Recently, the DMTF published preliminary specifications suggesting a method for exposing CIM resources with WS-Man [15,16]. Further study of those documents shows that WBEM and WS-Man are related technologies. By acting on the resource layer of a service-based management deployment, WebDMF is complementary to those approaches.

References

1. Stallings, W.: SNMP, SNMPv2, SNMPv3, RMON 1 and 2. Addison Wesley, Reading (1999)
2. CIM Infrastructure Specification. DMTF Standard, DSP0004 (2005)
3. Representation of CIM in XML. DMTF Standard, DSP0201 (2007)
4. CIM Operations over HTTP. DMTF Standard, DSP0200 (2007)
5. Oikonomou, G., Apostolopoulos, T.: WebDMF: A Web-based Management Framework for Distributed Services. In: The 2008 International Conference of Parallel and Distributed Computing (2008)
6. Zhou, S., Katto, J., Yasuda, Y.: Supporting Consistency Management in Dynamic Content Distribution Overlays. In: The Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services. IEEE Computer Society, Los Alamitos (2005)
7. Verma, D.C., Calo, S., Amiri, K.: Policy-based Management of Content Distribution Networks. IEEE Network 16(2), 34–39 (2002)
8. Bivens, A., Gupta, R., McLean, I., Szymanski, B., White, J.: Scalability and Performance of an Agent-based Network Management Middleware. International Journal of Network Management 14, 131–146 (2004)
9. Fortino, G., Russo, W.: Using p2p, Grid and Agent Technologies for the Development of Content Distribution Networks. Future Generation Computer Systems 24(3), 180–190 (2008)
10. Guillaume, P., van Steen, M.: Globule: A Collaborative Content Delivery Network. IEEE Communications Magazine 44(8), 127–133 (2006)
11. OpenCDN Project, `http://labtel.ing.uniroma1.it/opencdn/`
12. Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostic, D., Chase, J., Becker, D.: Scalability and Accuracy in a Large-Scale Network Emulator. In: 5th Symposium on Operating Systems Design and Implementation (OSDI) (2002)
13. Web Services for Management (WS Management). DMTF Specification, DSP0226 (2006)
14. An Introduction to WSDM. OASIS committee draft (2006)
15. WS-CIM Mapping Specification. DMTF Preliminary Specification, DSP0230 (2006)
16. WS-Management CIM Binding Spec. DMTF Preliminary Specification, DSP0227 (2006)
17. Common Manageability Programming Interface (CMPI), The Open Group Technical Standard (2004)

Crawling Bug Tracker for Semantic Bug Search

Ha Manh Tran, Georgi Chulkov, and Jürgen Schönwälder

Computer Science, Jacobs University Bremen, Germany
{h.tran,g.chulkov,j.schoenwaelder}@jacobs-university.de

Abstract. The Web has become an important knowledge source for resolving system installation problems and for working around software bugs. In particular, web-based bug tracking systems offer large archives of useful troubleshooting advice. However, searching bug tracking systems can be time consuming since generic search engines do not take advantage of the semi-structured knowledge recorded in bug tracking systems. We present work towards a semantics-based bug search system which tries to take advantage of the semi-structured data found in many widely used bug tracking systems. We present a study of bug tracking systems and we describe how to crawl them in order to extract semi-structured data. We describe a unified data model to store bug tracking data. The model has been derived from the analysis of the most popular systems. Finally, we describe how the crawled data can be fed into a semantic search engine to facilitate semantic search.

Keywords: Bug tracking system, Bug crawler, Semantic search.

1 Introduction

Trouble ticket systems and bug tracking systems are widely deployed in the information technology industry. Software and hardware companies use bug tracking systems during the development cycle to track bugs and design issues, or during later phases of the product lifecycle to keep track of defect reports and to obtain quality indicators. Almost all large open source projects maintain online bug tracking systems. In addition, there are many bug tracking systems supporting people who package open source software components for various software distributions. Some companies provide special online support forums (also known as communities or knowledge bases) for their products that often resemble bug tracking systems.

The fast growing amount of online information that can be used to resolve problems has led to a situation where system administrators and network operators often use search engines in order to find hints how to resolve a specific problem. However, it is our experience that searching in this way is not as efficient as we would like it to be; generic search engines do not seem to take advantage of the data found in trouble ticket or bug tracking systems.

Trouble ticket systems and bug tracking systems contain semi-structured data. Predefined fields are used to keep track of the status and metadata associated with a problem report while textual descriptions are used to describe the problem and to document the process for resolving the problem. Exploiting this semi-structured data has been considered a challenge. An early study in [1] suggests avoiding textual descriptions as much as possible since they cause difficulties in processing trouble tickets automatically. While this recommendation makes sense from a programmer's perspective, it clearly does not match the requirements of users who prefer to write down free-form text. Other studies [2,3] exploit only predefined fields that only use binary, numeric or symbolic values, an approach that has several limitations. These works have experimented with using trouble tickets as a basis for case-based reasoning (CBR) systems.

The semi-structured data contained in bug tracking systems is a valuable resource. It can be used to construct specialized search systems that have access to and knowledge of metadata out of reach to general text-based search engines. It can also be used to build automated reasoning systems that help to diagnose problems based on past experience. The goal of the work presented in this paper is to build a semantics-based bug search system and a bug dataset that can be used by a distributed case-based reasoning system which we are developing [4,5]. The aim of our system is to find relevant information quickly and to cope with the fact that the relevance of bug records changes quickly, due the short lifecycles of today's software products and services. Our contribution in this paper is fourfold:

1. We present a study of popular bug tracking systems and their features that can be used by semantic bug search or automated reasoning systems.
2. We present two methods to crawl bug tracking systems and to extract data.
3. Based on an analysis of the data models used by existing bug tracking systems, we develop a unified data model to store bug tracking data.
4. Finally, we apply a multi-vector representation (MVR) [5] to bug reports in order to enable semi-structured bug data search on the bug database.

The rest of the paper is structured as follows: In Section 2, we provide a study of some popular bug tracking systems. Section 3 explains how bug data can be extracted from these systems and Section 4 describes the unified data model we have developed to store the extracted bug data. Section 5 explains how the data are used for finding similar bugs and provides an experimental evaluation. We discuss related work in Section 6 before we conclude the paper in Section 7.

2 Bug Tracking Systems

Trouble ticket systems (TTS) have been widely used by network operators in order to assure the quality of communication services. A bug tracking system (BTS) is a special trouble ticket system used to keep track of software bugs. BTSs in general aim to improve the quality of software products. They do so by keeping track of current problems, and maintaining historical records of previously experienced issues. They also establish an expert system that allows to search

Table 1. Overview of bug tracking systems and some of their features (as of October 2007). Items marked with * are optional for each site. † reads dependencies.

Tracker	License	Access	Updates	Schema	Dep.†	Search
Bugzilla	MPL	HTML,XML-RPC*	SMTP, RSS*	textual	optional	filter,keywords
Mantis	GPL	HTML, SOAP*	SMTP, RSS	graphical	yes	filter
Trac	BSD	HTML	SMTP, RSS*	graphical	no	filter,keywords
Debian BTS	GPL	HTML, SMTP	SMTP	unknown	optional	filter
phpBugTracker	GPL	HTML	SMTP	textual	yes	filter,keywords
Flyspray	LGPL	HTML	SMTP,RSS,XMPP	unknown	yes	filter,keywords

for similar past problems, and provide reports and statistics for performance evaluation of the services [6].

We explore the features of several BTSs, focusing on several properties that are important for obtaining data from them. For each BTS we checked whether the BTS supports the functionality in question, and if so, which bug sites (i.e., specific installations of a BTS) support that function. A small sample of our results for BTSs and sites is given in Table 1 and Table 2 respectively. An explanation of each column in these tables is given below.

For each BTS, we looked at how the system is licensed for use, how its collection of bugs can be accessed, whether it is possible to easily receive notification of updated data, whether the database schema used by the BTS was available, whether the BTS keeps track of bug dependencies, and whether the BTS can be searched for bugs with a given property.

The license of a BTS affects its popularity. Generally, large free software projects tend to prefer a BTS licensed as free software itself. These projects also tend to be the ones that make their BTS sites public. For this reason, we excluded proprietary BTSs from our study.

The most important function of a BTS is retrieving bug reports in their most current states. All systems must at a minimum have an HTML-based web interface, but convenient automated retrieval requires a formalized programmatic interface, either based on XML-RPC or SOAP. While all BTSs support e-mail (via SMTP) as an update notification mechanism, e-mail is non-trivial to use for a web application. For instance, to receive notifications for all bug reports in a BTS by e-mail, an application would need to have its own e-mail address, register an account with that e-mail address in the BTS, and subscribe for every bug report of interest. Some systems support RSS or Atom feeds, which are significantly easier to use by a program.

In order to understand the structure of the information stored in a BTS, we investigated whether the underlying data model is documented. Some systems provide this information in a textual format while others provide graphical representations, usually in ad-hoc notations. Some systems do not provide a clear description of the data model underlying the BTS and it is necessary to reverse engineer the data model by looking at concrete bug reports.

Tracking any dependency relations between bug reports is useful, because it helps to correlate bugs. Generally, a bug is dependent on another bug if it cannot be resolved or acted upon, until the dependency is itself resolved or acted upon.

Table 2. Some popular bug tracking sites (as of October 2007). A plus indicates that we were unable to get precise numbers and our numbers present a lower bound. † reads dependencies.

Site	System	Version	Bugs	Activity	Custom	RPC	RSS	Dep.†
bugs.debian.org	Debian BTS	N/A	349346	1036	N/A	no	no	no
bugs.kde.org	Bugzilla	unknown	9655+	24+	light	no	no	no
bugs.eclipse.org	Bugzilla	unknown	204600	746	heavy	yes	yes	yes
bugs.gentoo.org	Bugzilla	unknown	183365	538	none	no	yes	yes
bugzilla.mozilla.org	Bugzilla	3.0.1+	173885	721	none	yes	yes	yes
bugzilla.redhat.com	Bugzilla	2.18-rh	177724	unknown	light	yes	yes	yes
qa.netbeans.org	Bugzilla	unknown	116639+	unknown	heavy	no	no	yes
bugs.digium.com	Mantis	unknown	10765	63	none	no	yes	yes
bugs.scribus.net	Mantis	1.0.7	6142	24	none	no	yes	yes
bugtrack.alsa-project.org	Mantis	1.0.6	3430	22	none	no	no	yes
dev.rubyonrails.org	Trac	0.10.5dev	11493+	unknown	none	no	yes	no
trac.edgewall.org	Trac	unknown	5948	unknown	none	no	yes	no
bugs.icu-project.org	Trac	0.10.4	5845	unknown	none	no	yes	no

Some systems allow full keyword search for their reports, while others only support searching via a set of predefined filters applied on the entire bug database. The former is more useful for an automated system that aims to provide keyword search capabilities itself.

Based on popularity and available documentation, we chose to focus on Bugzilla, Mantis, Trac and Debian BTS. With the exception of the Debian BTS, which is only used for the Debian operating system, the BTSs we chose all publish lists of known public sites that use them for bug tracking. Starting from these lists, we investigated all sites that are accessible and did not require authentication to browse their repositories (about 85 sites). Table 2 lists the sites with the largest number of bugs for each BTS. For each site, Table 2 shows which version of what BTS is used, how many bugs are stored there in total and how many have been added in one week, indicating the activity of the site. The table also specifies whether the site has been customized from its base BTS, whether it supports a programmatic XML-RPC interface or RSS feeds, and whether the site supports bug dependency relations.

The version of the underlying BTS largely impacts the set of available features. For example, Bugzilla only supports RSS feeds as of version 2.20, and XML-RPC as of version 3.0. Note that some sites hide this version number, possibly because this information may be sensitive with respect to security exploits in the BTS source code. The number of stored bugs and the rate of opening new bugs indicate the popularity and activity of a site. The margin between the most popular Bugzilla sites and the most popular sites using other BTSs is very large. We believe that the reason is that Bugzilla was the first widely-known open-source BTS when it was released in 1998. Mantis was only started in late 2000, and Trac is even newer.

Some sites customize their BTS in order to provide better integration of the bug tracker into the rest of their web site. While some sites only change the visual appearance of the BTS (marked as "light" customization in Table 2), others also modify the functionality of the BTS (marked as "heavy" customization). Customized sites pose a problem for automated bug retrieval: a system that is designed to derive structured data from presentational HTML (see Section 3.2) will generally fail to handle a significant change of a site's appearance. In addition, customizing a BTS naturally makes upgrading the site to a newer version of the BTS much more difficult; therefore customized sites tend to lag behind in version number, and consequently lack features such as RSS feeds or XML-RPC support.

Programmatic interfaces provided by protocols like XML-RPC or SOAP can be used by programs to directly query a bug tracker for structured data, without having to guess the value of any fields presented in human-readable form (HTML, SMTP). While this greatly simplifies interfacing to that bug tracker, no BTS currently makes such an interface a default option. It is an optional feature of the BTS at best, and not supported at all at worst. Only very few sites actually deploy and enable such programmatic interfaces, and clearly relying on their availability is not sufficient. RSS, on the other hand, is much more widely supported. RSS feeds allow programs to query a bug tracker for any updated bug reports, and while they are not as useful as XML-RPC interfaces, they still provide a better alternative to SMTP update notification.

3 Retrieving Data from BTSs

This section describes ways to retrieve semi-structured data from BTSs. First, we describe how we can exploit application programming interfaces (APIs) provided by the BTSs themselves. Since such APIs are only available on some sites, we have also implemented a web crawler specialized for bug tracking systems.

3.1 Exploiting APIs

The Bugzilla BTS provides an XML-RPC web service interface for users to access and modify bug reports. Users can write an external tool to interact with Bugzilla through several web service modules: The *User* module allows applications to create user accounts and to log in/out using an existing account. The *Bug* module can be used to file a new bug in Bugzilla, or to get information about already filed bugs. The *Product* module allows applications to list the available Products and to get information about them. Products are Bugzilla's top-level categories for bugs. Finally, the *Bugzilla* module provides functions to retrieve some general information about a Bugzilla installation.

The Bugzilla XML-RPC API in addition allows programs to retrieve a large number of bug reports in a single request using an array of bug identifiers. This saves much time when downloading many bug reports from a single Bugzilla website that supports XML-RPC.

We have implemented a crawler that uses several methods provided by the Bugzilla web service interface. It acts like an XML-RPC client that submits a bug identifier to a Bugzilla server and obtains the details of the bug (i.e., a list of field-value pairs) from the Bugzilla server. Note that the XML-RPC API is rarely available on production BTS installations (see Section 2) and is incomplete: it

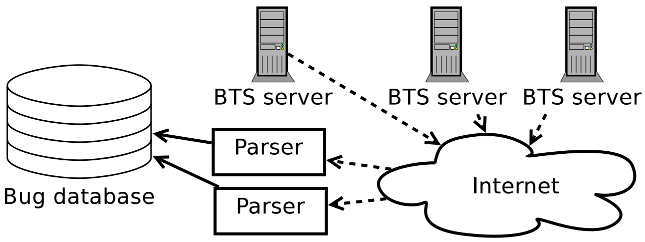


Fig. 1. Architecture of the Buglook crawler

restricts users from retrieving any attachments from a bug report, as well as the bug description or any related discussion entries. We access this information via another tool (described below).

Unlike all other BTSs, the Debian BTS allows users to access the raw bug data directly. Users can use the *rsync* utility to copy the whole bug database from bugs-mirror.debian.org. Debian's database is split into three sections: *bts-spool-db* for the active bug report spool, *bts-spool-archive* for bug reports that have been closed for a while and thus archived, and *bts-spool-index* for the bug index files. Each bug report is stored in four different files whose names consist of the bug number and either .log, .report, .status, or .summary as an extension.

3.2 Crawling with Buglook

While some BTSs provide a machine-readable web service interface to their bug data, most do not. In all systems where such an interface is supported, it is an optional feature, and because optional features require additional effort from an administrator to be set up, they are rarely available. In addition, a web service interface often provides much less data than the human-readable web interface that is most commonly used. Clearly, relying on the availability of a web service API is unrealistic. To solve this problem, we created Buglook [7], a tool which attempts to directly use the presentational HTML-based web interface in order to get as much access to information as ordinary users.

The problem with presentational HTML pages is that the same structure can be presented in vastly different ways. As an example, consider an algorithm that must detect the end of a bug report comment and the beginning of the next one. In HTML, this boundary could be encoded as a closing `<div>` tag and the opening of another `<div>`, or as a new paragraph (`<p>`), or why not a sequence of newlines (`<br>`)? There is nothing preventing the same elements from being used in another context, while being rendered differently (dictated by CSS tags). No consistency can be expected.

To tackle this problem, we note the following: (i) Because bug report pages are generated from a template, all bug reports within a single BTS site have the same structure. An algorithm that can parse one bug report can parse all bugs in that site. (ii) BTS sites that use the same software have similar bug structure, and often similar appearance. The underlying BTS software determines the structure of the data it can work with, and only allows presentational customization of the displayed HTML pages. (iii) For each BTS, there is a canonical appearance. In general, most sites do not find it necessary to customize their appearance, and use the one that the BTS provides by default.

Buglook (Figure 1) uses a small set of parsers defined for each BTS's canonical appearance, together with specialized parsers for the most important customized sites. These parsers, called "site modules", can provide a very high degree of coverage of all BTS sites. The set of sites covered by a site module is its "sitetype". Sitetypes are essentially equivalence classes of BTS sites, with respect to parsing.

Buglook's bug extraction component consists of two essential parts - the set of site modules, and a common component independent of all of them. The common component is responsible for generic tasks such as downloading web pages, parsing HTML, etc. The site modules encapsulate all logic unique to a given sitetype. This distinction allows more sitetypes to be supported with minimal duplication of effort. To support a sitetype, a site module must implement a fixed interface to the common component.

4 Unified Data Model

In order to integrate bug data from different BTSs into a single bug database, we define a unified data model for bugs. This model must be simple and easy to use for our purpose; it is not necessary to be able to represent all available details of all systems. Our investigation of the database schemas of the four BTSs we considered exposes several interesting observations. The bug formats of Bugzilla, Trac and Mantis share many similar fields that can be classified in two main groups:

1. The administrative metadata associated with a bug is often represented as field-value pairs with very precise semantics, such as *id*, *severity*, *reporter*, *summary*, among others.
2. The descriptions detailing the bug and any followup discussion or actions are typically represented as free-form (i.e., non-formal) textual attachments.

Because the unified data model is used to support semantic search, we aim to extract fields from bug reports in such a way as to minimize the loss of bug information. We introduce new fields that establish the relationships between bugs or provide for more sophisticated classification. The values of these fields can be derived differently for each BTS: Mantis users can manually specify the

Table 3. Severity of bugs

Unified model	Bugzilla	Trac	Mantis	Debian
critical	blocker, critical	blocker, critical	block, crash	critical, grave, serious
normal	major	major	major	important, normal
minor	minor, trivial	minor, trivial	minor, tweak, text, trivial	minor
feature	enhancement	-	feature	wishlist

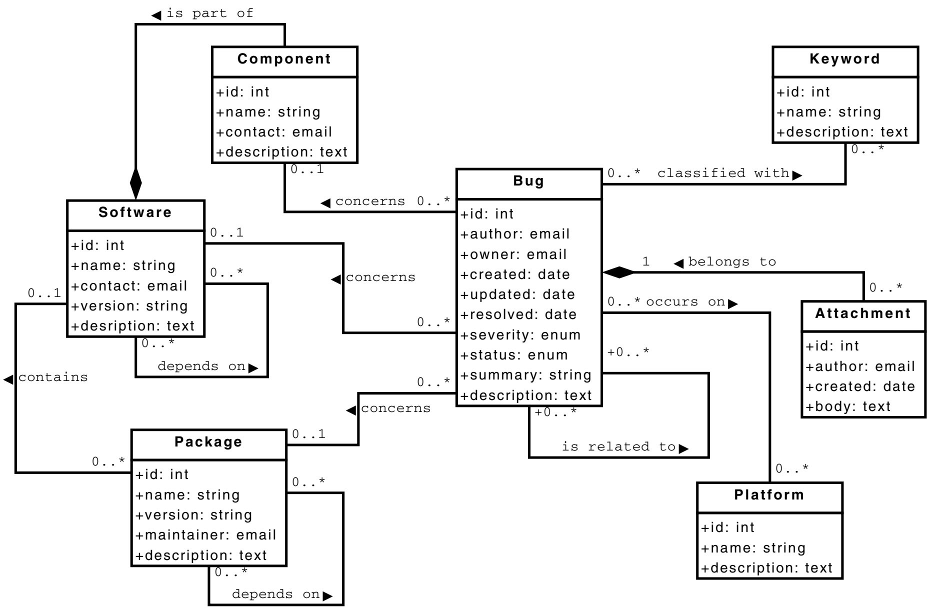


Fig. 2. Unified bug data model represented as a UML diagram

relation of a bug to other bugs when reporting it; Debian users can indicate which package a bug relates to; and Bugzilla and Trac offer a keyword field that enables the classification of bugs. To exploit data from a bug's description and its attachments, we use several text processing techniques [8,9].

Figure 2 shows our unified bug data model in the form of a UML class diagram. The central class is the `Bug` class. The `id` attribute of a `Bug` instance uniquely identifies a bug. We use the URL where a bug can be retrieved as its identifier. Most of the attributes of a `Bug` instance can be easily extracted from the retrieved data. Our `severity` attribute is probably the most interesting to fill correctly, because BTSs have very different severity classifications for bugs. Table 3 shows how we map the severity values of the BTSs into our data model, which only distinguishes the severity values *critical*, *normal*, *minor*, and *feature*. The `status` attribute of a `Bug` instance only has two values: the value *open* represents what BTSs call *unconfirmed*, *new*, *assigned*, *reopened* bugs while the value *fixed* represents what BTSs call *resolved*, *verified*, and *closed* bugs.

Free-form textual descriptions are modelled as `Attachment` objects. Every `Attachment` belongs to exactly one `Bug` object. Some BTSs provide information about the platforms affected by a bug. We represent platforms (such as "Windows2000" or "MacOS X") as `Platform` objects. The `Keyword` class represents keywords used to describe and classify bugs.

The left part of Figure 2 models what piece of software a bug is concerned with. While some BTSs are only concerned with bugs in a specific piece of software, software in larger projects is split into components and bugs can be related to specific components. The classes `Software` and `Component` model this structure. The Debian BTS is somewhat different from the other BTSs as it is primarily used to track issues related to software “packages”, that is software components packaged for end user deployment. Since there is a large amount of meta information available for Debian software packages (dependency, maintainer and version information), we have introduced a separate `Package` class to represent packaged software.

5 Semi-structured Bug Data Search

BTSs only support keyword search and restricted meta-data search by predefined fields and values (as discussed in Section 2); e.g., searching for bugs with the *resolved* status or bugs with the *critical* severity. This section presents the performance of different search algorithms on semi-structured bug data from the unified dataset.

We have previously evaluated the combination of fulltext search and meta-data search, namely `ft-md` search, on the CISI and MED bibliographic datasets whose documents contain semi-structured data [5]. With the bug dataset, meta-data search exploits significant keywords extracted from bug contents, such as type of problems, scope of problems, typical symptoms, error messages and distinct terms. A set of keywords is represented by a field-value vector. The similarity of two field-value vectors is estimated by the sum of weight values of matched keywords. Fulltext search involves indexing terms from bug contents using text processing techniques [8,9]. Each bug is converted to a term vector which is then transformed to a real number vector (or a semantic vector) using algebraic computation. The similarity of two semantic vectors is evaluated by the cosine of these vectors.

We consider keyword search as baseline search that evaluates the similarity between a bug and a query by simply matching keywords from the query to the bug content without considering the significance of keywords. We use a *matching rate* metric to compare the performance of the combination of search algorithms: `ft-md` combining fulltext search and meta-data search, `ft-bl` combining fulltext search and keyword search, and `md-bl` combining meta-data search and keyword search. The matching rate r is the ratio of the number of the identical bugs obtained by two search algorithms to the minimum number of bugs obtained by these algorithms for a query:

$$r = \frac{|S_x \cap S_y|}{min(N_x, N_y)} \tag{1}$$

where S_x and S_y are the resulting set of search algorithms x and y per query, $|S|$ is the size of set S, and parameters N_x and N_y are the total number of bugs obtained by search algorithms x and y per query. Intuitively, if two search algorithms are good, the probability of a large number of identical bugs obtained by these algorithms is high. This metric is more feasible and flexible than the recall rate or precision rate metrics that require knowledge of the correct number of relevant bugs per query. Note that it is difficult to obtain this number from a new and large dataset.

The evaluation of a semantic bug search engine on a large dataset of several hundred thousand bugs will be reported in another study. In these experiments, the dataset contains 11.077 bugs, and the number of obtained bugs N_{ft}, N_{md} and N_{bl} are set to 100 by sorting the resulting sets according to the similarity value and selecting only the top N elements. A set of 50 queries include pieces of textual descriptions, distinct keywords, typical symptoms and error messages that are extracted from bug contents. Terms or keywords from bug contents are stemmed by the Porter stemming algorithm [10] and weighted by the term frequency-inverse document frequency (`tf-idf`). Semantic vectors are generated by computing singular value decomposition using the single-vector Lanczos algorithm [11] implemented in `svdlibc`. The experiments were performed on an x86_64 GNU/Linux machine with two dual-core AMD Opteron(tm) processors running at 2 GHz with 4 GB RAM.

The left plot shown in Figure 3 reports the average matching rate of `ft-md`, `md-bl` and `ft-bl` over an increasing number of queries. The `ft-md` line stays at at a matching rate of 0.3 on average and reaches 0.33 finally, whereas the `md-bl` and `ft-bl` lines start lowly and reach 0.22 and 0.12, respectively. `Ft-md` found more identical bugs than other combinations. Furthermore, `md-bl` obtained better results than `ft-bl`. The results of the last two combinations are relatively different, illustrating that `bl` search obtains less consistent and reliable results. `Bl` search combines better with `md` search than with `ft` search because `md` and `bl` both use keyword comparison to estimate similarity.

The query distribution for `ft-md` shown in the right plot in Figure 3 indicates that more than 50% of the queries receives a matching rate higher than 0.3. These queries tend to focus on specific distinct characteristics of bugs, whereas the other queries tend to be more general or vague, resulting in a large number of improper bugs obtained.

We relax the number of obtained bugs for only `bl` search, namely `rlbl`; N_{rlbl} is set to 500. The left plot shown in Figure 4 shows that `md-rlbl` improves the

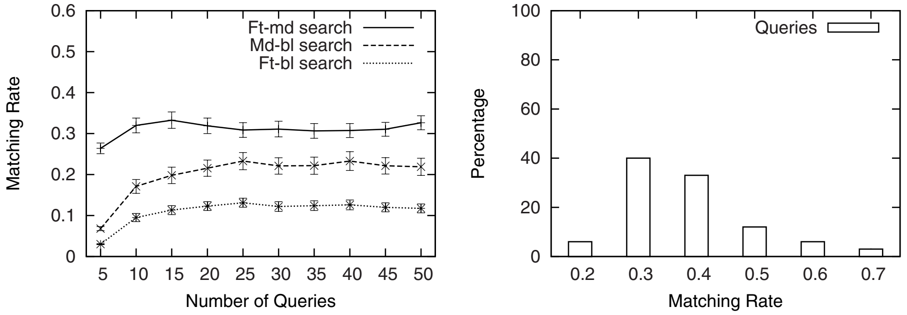
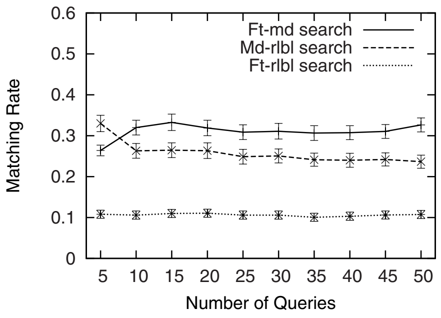


Fig. 3. Average matching rate by number of queries for `ft-md`, `md-bl` and `ft-bl` (left). Query distribution by matching rates for `ft-md` (right).



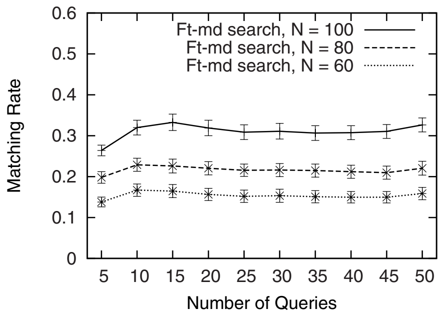


Fig. 4. Average matching rate by number of queries for `ft-md`, `md-rlbl` and `ft-rlbl` with a large number of bugs obtained by baseline search (left). Average matching rate by number of queries with various numbers of bugs obtained by `ft-md` (right).

matching rate to 0.25 on average, whereare `ft-rlbl` remains unchanged. `Rlbl` search and `md` search finds more identical bugs than `rlbl` search and `ft` search. While `md` search is similar to `rlbl` search, it is much different from `ft` search in the way of measuring similar bugs. Combining `ft` search and `md` search, therefore, works well for semi-structured bug data.

We investigate further `ft-md` search by restricting the number of obtained bugs; N_{ft} and N_{md} are both set to 100, 80 and 60. The right plot in Figure 4 indicates that the matching rate reduces when the number of obtained bugs reduces. Note that bugs are chosen by their ranking. The ranking of obtained bugs are different between `ft` search and `md` search. This is caused by three reasons: first, while BTSs may contain duplicated bugs, the number of these bugs is small, thus the number of truly similar bugs is also small; second, the ranking values of obtained bugs contain errors from indexing and ranking bugs, especially when a query is general, ranking many similar bugs is affected by these errors; last, as described above, ineffective queries make results inaccurate. These reasons also cause the low matching rate. However, since the bug dataset is wide and diverse in scope, we believe that `ft-md` search achieving an average matching rate of 0.3 is reasonable.

6 Related Work

The X.790 recommendation from ITU-T [12], the RFC 1297 from the IETF [13] and the NMF501 and NMF601 documents from the TMF [14,15] define terminology and basic functions for reporting and managing trouble tickets. NetTrouble [16] introduces advanced features for trouble ticket systems (TTSs) that include a distributed database concept for geographic dissemination, an administrative domain concept for the multi-organizational characteristics of network management environments, and an administrative model for hierarchical decomposition. The work in [17] proposes a generic interface and a generic data structure, namely customer service management trouble report, to support inter-domain problem management between customer and service provider. Our work towards a unified data model is to some extend related to these efforts. However, instead of designing a feature rich data model from scratch, we took the opposite approach to extract the common core from the data models used by existing systems.

Since the problem-solving knowledge in TTSs can be exploited to search for similar problems or infer typical solutions, several studies discussed in [1] have used TTSs associated with artificial intelligence techniques for finding and resolving similar problems. A study in [2] has proposed a CBR system to resolve network problems by retrieving similar problems and adapting solutions to novel problems. Trouble tickets obtained by a TTS are used as cases for evaluating the system. The DUMBO system [3] also takes advantage of TTSs to propose solutions for network problems. This system provides six types of features to represent trouble tickets, and employs similarity and reliability measurement for proposing solutions. The main limitations of these systems, however, contain the inexpressive representation of trouble tickets and the lack of trouble ticket sources [1]. We consider these issues in this work by using the unified data model that allows various bug reports to be collected in one bug database, and by applying MVR to bug reports to enable semantic search on the bug database.

7 Conclusions

We have provided a study of existing BTSs with a specific focus on the four most popular open source systems, namely Bugzilla, Trac, Mantis and the Debian bug tracking system. Widely used public BTSs based on these software systems contain a large number of bug reports that can be used for building bug datasets. Such datasets are invaluable for evaluating systems such as case-based reasoning engines or semantic search engines. We have used web service APIs and a special purpose web crawler (Buglook) to obtain a large number of bug reports from several large BTSs. In order to store the data in an effective way, we have developed a unified bug data model that is able to capture the most important aspects of the data maintained by the various BTSs we have analyzed. Our model enables interoperable aggregation of data from different sources, useful for various purposes ranging from efficient wide-scale search to automated reasoning systems.

The multi-vector representation method (MVR) [5] has been used to perform semantic search experiments on the unified bug dataset. MVR exploits semi-structured bug data to search for similar bugs with salient features. The experimental results indicate that (i) the combination of fulltext search and meta-data search (using MVR) outperforms the other combinations of fulltext search and baseline search or of meta-data search and baseline search, (ii) baseline search provides less consistent and reliable results, and (iii) the bug dataset is wide and diverse in scope.

We are currently implementing a complete online semantic search system that will accept user queries so that a larger number of users can test and evaluate our system. This system also allows us to evaluate search latency and bug synchronization on a large bug dataset. Future work involves extending the unified data model to support another semantic bug search system, where bug reports are represented in the resource description framework (RDF). In addition, refined and unified datasets are used to evaluate the problem-solving capability of our distributed case-based reasoning system. Such systems will certainly be a practical tool for anyone who needs to troubleshoot a software system with a public bug tracking system.

Acknowledgments. The work reported in this paper is supported by the EC IST-EMANICS Network of Excellence (#26854).

References

1. Lewis, L., Dreo, G.: Extending Trouble Ticket Systems to Fault Diagnostics. IEEE Network Special Issue on Integrated Network Management 7(6), 44–51 (1993)
2. Lewis, L.: A Case-Based Reasoning Approach to the Resolution of Faults in Communication Networks. In: Proc. 3rd International Symposium on Integrated Network Management (IM 1993), pp. 671–682. North-Holland, Amsterdam (1993)
3. Melchiors, C., Tarouco, L.: Fault Management in Computer Networks Using Case-Based Reasoning: DUMBO System. In: Althoff, K.-D., Bergmann, R., Branting, L.K. (eds.) ICCBR 1999. LNCS (LNAI), vol. 1650, pp. 510–524. Springer, Heidelberg (1999)
4. Tran, H.M., Schönwälder, J.: Distributed Case-Based Reasoning for Fault Management. In: Bandara, A.K., Burgess, M. (eds.) AIMS 2007. LNCS, vol. 4543, pp. 200–203. Springer, Heidelberg (2007)
5. Tran, H.M., Schönwälder, J.: Fault Representation in Case-Based Reasoning. In: Clemm, A., Granville, L.Z., Stadler, R. (eds.) DSOM 2007. LNCS, vol. 4785, pp. 50–61. Springer, Heidelberg (2007)
6. Bloom, D.: Selection Criterion and Implementation of a Trouble Tracking System: What's in a Paradigm? In: Proc. 22nd Annual ACM SIGUCCS Conference on User Services (SIGUCCS 1994), pp. 201–203. ACM Press, New York (1994)
7. Chulkov, G.: Buglook: a search engine for bug reports. Seminar Report. Jacobs University Bremen (May 2007)
8. Deerwester, S., Dumais, S., Landauer, T., Furnas, G., Harshman, R.: Indexing by Latent Semantic Analysis. Journal of the Society for Information Science 41(6), 391–407 (1990)
9. Berry, M.W., Drmac, Z., Jessup, E.R.: Matrices, Vector Spaces, and Information Retrieval. SIAM Review 41(2), 335–362 (1999)
10. Porter, M.F.: An algorithm for suffix stripping. Readings in Information Retrieval, 313–316 (1997)
11. Golub, G.H., Underwood, R.: The block lanczos method for computing eigenvalues. Mathematical Software III, 361–377 (1977)
12. ITU-T. Trouble Management Function for ITU-T Applications. X.790 Recommendation (1995)

13. Johnson, D.: NOC Internal Integrated Trouble Ticket System Functional Specification Wishlist. RFC 1297 (1992)
14. TMF. Customer to Service Provider Trouble Administration Business Agreement. NMF 501, Issue 1.0 (1996)
15. TMF. Customer to Service Provider Trouble Administration Information Agreement. NMF 601, Issue 1.0 (1997)
16. Santos, L., Costa, P., Simes, P.: NetTrouble: A TTS for Network Management. In: ITS 1998, pp. 480–485. IEEE Computer Society, Los Alamitos (1998)
17. Langer, M., Nerb, M.: Defining a Trouble Report Format for the Seamless Integration of Problem Management into Customer Service Management. In: Proc. 6th Workshop of the OpenView University Association (OVUA 1999) (1999)

A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans

Weverton Luis da Costa Cordeiro[1], Guilherme Sperb Machado[1], Fabrício Girardi Andreis[1], Alan Diego Santos[1], Cristiano Bonato Both[1], Luciano Paschoal Gaspary[1], Lisandro Zambenedetti Granville[1], Claudio Bartolini[2], and David Trastour[3]

[1]Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
[2]HP Laboratories Palo Alto, USA
[3]HP Laboratories Bristol, UK
{weverton.cordeiro, gsmachado, fgandreis, adsantos, cbboth, paschoal, granville}@inf.ufrgs.br, {claudio.bartolini, david.trastour}@hp.com

Abstract. Change design is one of the key steps within the IT change management process and involves defining the set of activities required for the implementation of a change. Despite its importance, existing approaches for automating this step disregard the impact that actions will cause on the affected elements of the IT infrastructure. As a consequence, activities that compose the change plan may not be executable, for example, due to runtime constraints that emerge during the change plan execution (*e.g.*, lack of disk space and memory exhaustion). In order to address this issue, we propose a solution for the automated refinement of runtime constraint-aware change plans, built upon the concept of incremental change snapshots of the target IT environment. The potential benefits of our approach are (*i*) the generation of accurate, workable change plans, composed of activities that do not hinder the execution of subsequent ones, and (*ii*) a decrease in the occurrence of service-delivery disruptions caused by failed changes. The experimental evaluation carried out in our investigation shows the feasibility of the proposed solution, being able to generate plans less prone to be prematurely aborted due to resource constraints.

1 Introduction

The increasing importance and complexity of IT infrastructures to the final business of modern companies and organizations has made the Information Technology Infrastructure Library (ITIL) [1] the most important reference for IT service deployment and management. In this context, ITIL's best practices and processes help organizations to properly maintain their IT services, being of special importance to those characterized by their large scale and rapidly changing, dynamic services.

Among the several processes that compose ITIL, *change management* [2] plays an important role in the efficient and prompt handling of IT changes [3]. According to this process, changes must be firstly expressed by the *change initiator* using *Requests for Change* (RFC) documents. RFCs are declarative in their nature, specifying what should be done, but not expressing how it should be performed. In a subsequent step, an *operator* must sketch a preliminary *change plan*, which encodes high level actions that materialize the objectives of the RFC. Latter steps in this process include *planning*, *assessing and evaluating*, *authorizing and scheduling*, *plan updating*, *implementing*, and *reviewing and closing* the submitted change.

Change *planning*, one of the key steps in this process, consists in refining, either manually or automatically, the preliminary plan into a detailed, actionable workflow (also called actionable change plan in this paper). Despite the possibility of manually refining change plans, automated refinement has the potential to provide better results for the planning phase, since it (*i*) decreases the time consumed to produce such actionable workflows, (*ii*) captures the intrinsic dependencies among the elements affected by changes, and (*iii*) diminishes the occurrence of service disruptions due to errors and inconsistencies in the generated plans [4].

Since the inception of ITIL, there has been some preliminary research concerning the automated refinement of change plans. For example, important steps have been taken towards formalizing change-related documents [5], exploring parallelism in the execution of tasks [3], and scheduling of change operations considering the long-term impact on *Service Oriented Architecture* environments [6]. However, despite the progresses achieved in the field, proposed solutions for change planning only consider simple actions (installation, upgrade) and do not model the pre-conditions and effects of more complex actions. The pre-conditions could be of a technical nature, such as a memory requirement, or could impose constraints on the change process, for instance requiring authorization before executing a given task. Effects model how actions modify each element of the IT infrastructure (*e.g.*, adding memory into a server or modifying configuration parameters of a J2EE server). Without taking into account such considerations, the actionable workflow, when executed, may be prematurely aborted (*e.g.*, due to lack of resources), leading to service-delivery disruption and leaving the IT infrastructure in an inconsistent state.

To fill in this gap, we propose a solution for the automated refinement of change plans that takes into consideration the runtime constraints imposed by the target IT environment. In contrast to previous investigations, our solution focuses on the impact that already computed actions will cause on the IT infrastructure, in order to compute the subsequent ones. To this effect, we introduce in this paper the notion of *snapshots* of the IT infrastructure, as representations of the intermediate states that the IT infrastructure would reach throughout the execution of the change plan. As a result, the refined change plans generated by our solution will be less prone to prematurely termination, therefore reducing the occurrence of change-related incidents.

The solution proposed in this paper is evaluated through the use of CHANGELEDGE, a prototypical implementation of a change management system that enables the design, planning and implementation of IT changes. We have qualitatively and quantitatively analyzed the actionable workflows generated from several different preliminary plans, considering a typical IT scenario.

The remainder of this paper is organized as follows. Section 2 discusses some of the most prominent research in the field of IT change management. Section 3 briefly reviews the models employed to represent IT related information. Section 4 details our runtime constraint-aware solution for the automated refinement of IT change plans. Section 5 presents the results achieved using the CHANGELEDGE system. Finally, Section 6 concludes the paper with remarks and perspectives for future work.

2 Related Work

In the recent years, several research efforts have been carried out in the area of IT change design. In this section, we cover some of the most prominent investigations.

Keller *et al.* [3] have proposed CHAMPS, a system for automating the generation of change plans that explore a high degree of parallelism in the execution of tasks. Change planning and scheduling are approached as an optimization problem. Although the system is able to evaluate technical constraints in the planning and scheduling of changes, the scope is limited to Service Level Agreements and policies. Since fine-grained control of resource constraints was not the focus of the work, modifications on the infrastructure produced by the already processed tasks of the plan under refinement are not taken into account when computing the subsequent ones. As a consequence, the resulting change plans may not be executable in practice.

In a previous work [5], we have proposed a solution to support knowledge reuse in IT change design. Although the solution comprises an algorithm to generate actionable change plans, this algorithm also performs all the computations considering a static view of the IT infrastructure. Actually, it was out of the scope of that work, as a simplification assumption, to deal with runtime constraints in the refinement of change plans.

Despite not directly related with the problem addressed in this paper, some additional research efforts on change management published in the recent years merit attention. Dumitraş *et al.* [6] have proposed *Ecotopia*, a framework for change management that schedules change operations with the goal of minimizing service-delivery disruptions. In contrast to CHAMPS, Ecotopia optimizes scheduling by assessing the long-term impact of changes considering the expected values for *Key Performance Indicators*. Trastour *et al.* [7] have formulated the problem of assigning changes to maintenance windows and of assigning change activities to technicians as a mixed-integer program. The main difference between this work and Ecotopia is the fact that human resources are also taken into account. Sauvé *et al.* [8] have proposed a method to automatically assign priorities to changes, considering the individual exposure of each requested change to risks as its execution is postponed. Finally, in another previous work [9], we have introduced the concept of *atomic groups* in the design of change plans with the purpose of providing our end-to-end solution to IT change management with rollback support.

Although change management is a relatively new discipline, the area has been quickly progressing, as evidenced by the previously mentioned related work. Nevertheless, in the particular case of change planning, the existing solutions are severely lacking with respect to deployment feasibility and IT infrastructure predictability. In the following sections we envisage a solution to address these issues.

3 Building Blocks of the Proposed Solution

In order to support the automated refinement of change plans, it is of paramount importance to formalize the change-related documents. Actually, this was a major concern in our previous work [5], in which we proposed models to (*i*) characterize dependencies between the elements that compose the IT infrastructure, (*ii*) express information about software packages available for consumption by a change process, and (*iii*) express unambiguously the changes that must be executed on the managed infrastructure. In this section, we briefly review the models that materialize this formalization: *IT infrastructure* and *Requests for Change & Change Plan*.

The *IT Infrastructure* model is a subset of the Common Information Model (CIM) [10], proposed by the Distributed Management Task Force (DMTF). It allows the representation of computing and business entities comprising an organization, as well as the relationship among them. For the sake of legibility and space constraints, we present in Fig. 1 a partial view of the model.

The root class *ManagedElement* permits to represent any *Configuration Item* (CI) present in the IT infrastructure (*e.g.*, physical devices, computer and application systems, personnel, and services). Relationships such as associations, compositions, and aggregations, map the dependencies among the elements comprising the infrastructure. In addition, *Check* and *Action* classes in this model represent relevant information for managing the lifecycle of *software elements* (*e.g.*, software upgrade and application system installation/uninstallation).

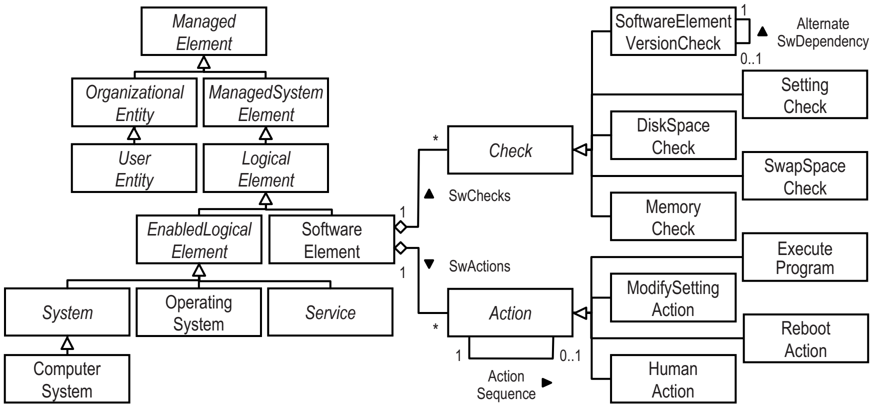


Fig. 1. Partial view of the IT Infrastructure model

Instances of class *Check* define conditions to be met or characteristics required by the associated software element for it to evolve to a new state (*e.g.*, *deployable*, *installable*, *executable*, or *running*). Possible checks include verification of software dependencies, available disk space and memory, and required environment settings. Each instance of class *Action*, in its turn, represents an operation of a process to change the state of the associated *SoftwareElement* (*e.g.*, from *installable* to *executable*). Examples of actions are invocation of a software installer/uninstaller, manipulation of files and directories, and modification of configuration files.

In addition to being used to represent the current IT infrastructure, the same model is also employed to define the *Definitive Media Library* (DML). The DML is a repository that specifies the set of software packages (along with their dependencies) that have been approved for use within the enterprise and that may be required throughout the change process.

In regard to the *Requests for Change & Change Plan* model, it enables the design of change-related documents and relies on both (*i*) guidelines presented in the ITIL Service Transition book [2], and (*ii*) the workflow process definition, proposed by the Workflow Management Coalition (WfMC) [11]. Classes such as *RFC* and *Operation* allow expressing the changes designed by the *change initiator*, while *ChangePlan*, *LeafActivity*, *BlockActivity*, *SubProcessDefinition*, and *TransitionInformation* enable the *operator* to model the preliminary plan that materializes the change. Please refer to our previous work [5] for additional information about this model.

4 Runtime Constraint-Aware Refinement of Change Plans

The models presented in the previous section represent the common ground for our runtime constraint-aware solution for automated refinement of IT change plans. In this section, we describe our solution by means of a conceptual algorithm, illustrated in Fig. 2.

In order to support our solution, we formalize a change plan C, in the context of this work, as a 4-tuple $\langle A, T, a_1, F\rangle$, where A represents the set of activities (or actions) $A = \{a_1, a_2, \ldots, a_n \mid n \in N \text{ and } n \geq 1\}$; T represents a set of ordered pairs of activities, called transitions, $T = \{l_1, l_2, \ldots, l_m \mid m \in N \text{ and } m \geq 1\}$; a_1 is the begin activity of the change plan ($a_1 \in A$); and F represents the set of end activities of the change plan ($F \subseteq A$). A transition $l = (a_i, a_j) \in T$ is directed from a_i to a_j, $\forall a_i, a_j \in A$, and may represent a conditional flow.

We denote our solution as a function $f(C, I, R) = C'$ (line 1), where C is the preliminary change plan; I represents the state of the IT infrastructure as in the instant in which the preliminary plan C is submitted for refinement; R represents the *Definitive Media Library* (DML); and C' represents the actionable workflow generated as a result of the refinement process.

As a first step towards the refinement, the submitted plan C is copied to C' (line 2), and the subset of unrefined activities contained in C is copied to A' (line 3). In a subsequent step (line 4), f creates an initial *snapshot* of the IT infrastructure, s_0. In the context of this work, we define snapshot as a representation of the differences between the current state of the IT infrastructure and the state it would reach after the execution of i activities contained in the change plan C ($0 \leq i \leq |A|$). These differences include, for example, newly installed (or removed) software, disk space and memory consumed (or freed), modified settings, and created (or deleted) files and directories (the dynamics of snapshots is further explained in Subsection 4.2). Considering that no new activities were added to the change plan C at the point s_0 is created, this step will yield a snapshot that describes no differences in comparison to the current state of the IT infrastructure.

As a last step, f invokes the execution of $f'(C', R, I, s_0, A')$ (line 5), which will actually perform the refinement process. We assume that C' is passed to f' by reference. Therefore, modifications performed to C' will be visible outside f'. After the execution of f', C' will be returned back to the *operator* (line 7), if refined (line 6). We consider a change plan C as refined if and only if, $\forall a \in A$, dependencies of a are already satisfied either by any $a_i \in A$ or by the current state of the IT infrastructure.

```
1   $f(C, R, I) = C'$ {
2     declare $C'$ = copy of the preliminary change plan $C$
3     declare $A'$ = set of unrefined activities from the preliminary change plan $C$
4     $s_0$ = initial snapshot of $I$, after the execution of 0 activities
5     $f'(C', R, I, s_0, A')$
6     if ($C'$ is refined)
7       return $C'$
8     else
9       return false
10  }
11
12  $f'(C, R, I, s_i, A)$ {
13    if ($A$ is empty)
14      return change plan $C$
15    else {
16      declare $X$: set of arrangements $Y$ of activities
17      $a_i$ = $i$-est activity $\in A$
18      declare $A' = A - \{a_i\}$
19      if ($a_i$ has no computable dependencies, given $I$, $s_i$, and $R$)
20        $f'(C, R, I, s_i, A')$
21      else {
22        $X$ = set of arrangements $Y$ of first level dependencies of $a_i$, given $I$, $s_i$, and $R$
23        for each $Y_i \in X$ {
24          declare $C' = C + Y_i$
25          declare $A'' = A' \cup Y_i$
26          $s_{i+1}$ = new snapshot of the IT infrastructure $I$, given $C'$, $I$, and $s_i$
27          $f'(C', R, I, s_{i+1}, A'')$
28        }
29      }
30    }
31  }
```

Fig. 2. Conceptual algorithm for runtime constraint-aware refinement of change plan

In case the plan returned by f' is not refined, the operator will receive a negative feedback (line 9). This feedback will mean that an actionable and executable workflow (for the preliminary plan C submitted) could not be achieved. Having this feedback, the operator could reformulate and resubmit the preliminary plan, therefore starting the refinement process over again.

Having presented a general view of our solution, in the following subsections we describe in more detail the recursive search for a refined change plan, and the concept of snapshots of the IT infrastructure.

4.1 Refinement of the Preliminary Change Plan

Function f' solves the problem of modifying the received preliminary plan C into an actionable workflow by using the *backtracking* technique [12]. This technique permits exploring the space of possible refinements for C, in order to build a refined plan that meets IT resource constraints. Fig. 3 illustrates the execution of f' using a simplified example. For the sake of clarity, only two levels of recursion are presented.

The preliminary plan C in Fig. 3 materializes an RFC to install an e-Commerce Web application, and is composed of the task *Install WebApp*. This task represents a *BlockActivity* derived from the set of actions necessary to install *WebApp* (arrow 1 in Fig. 3). The first verification performed by f' (line 13 in Fig. 2) is whether A, the set of activities that remain unrefined in the received plan C, is empty or not. If A is empty, C is returned back to f. Considering the example in Fig. 3, f' will receive in its first invocation (line 5) the set $A' = \{$*Install WebApp*$\}$.

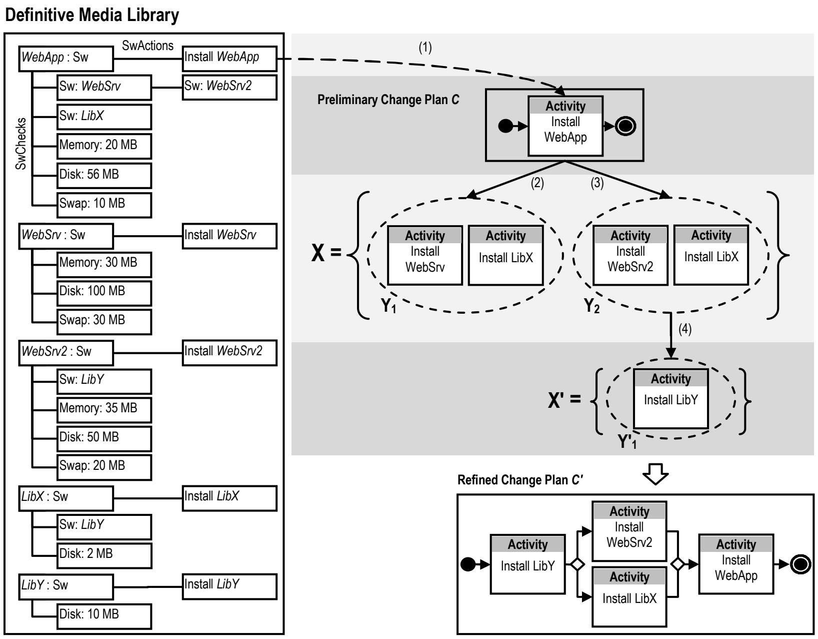


Fig. 3. Illustration of the functioning of f'

The algorithm f' starts by extracting an activity a_i from A (line 17), generating a new set A' (which contains all activities in A except a_i) (line 18). In our example, a_i is the activity *Install WebApp*, and the resulting A', an empty set. Subsequently, f' tests whether a_i has computable dependencies (line 19). An activity is said to have computable dependencies if: (*i*) the *Configuration Item* (CI) modified by a_i has checks (*SwChecks*) mapped in the DML and/or relationships in the IT repository (*e.g.*, shutting down service *Service*$_1$ requires shutting down *Service*$_2$ and bringing up *Service*$_3$), and (*ii*) the aforementioned dependencies (or checks) are not yet fulfilled in neither the current state of the IT infrastructure nor the current snapshot.

If a_i has no computable dependencies (*i.e.*, if all pre-conditions for the execution of a_i are already satisfied in either the IT or the current snapshot), f' invokes itself recursively (line 20), in order to refine another activity of the resulting A'. Otherwise, f' computes the set of arrangements of *immediate dependencies* (or *first level dependencies*) that (*i*) fulfill the pre-conditions for the execution of a_i, and (*ii*) would be executable in the current snapshot (considering the requirements of these arrangements). The arrangements returned from this step will be stored in X (line 22). In this set, Y_i represents each of the arrangements.

In our example, *Install WebApp* has two computable dependencies described in the DML: a web server (either *WebSvr* or *WebSrv2*) and a generic library (*LibX*). Therefore, the computation of X (line 22) yields a set containing two arrangements of possible immediate dependencies for a_i. The first is Y_1 = {*Install WebSrv*, *Install LibX*}, and the second is Y_2 = {*Install WebSrv2*, *Install LibX*}.

After that, f' searches for an arrangement Y_i in X that leads to a refined change plan (line 23). Although more than one Y_i may lead to a solution, the first Y_i to be tested will compose the refined plan. Considering the example, the first set tested was Y_1 (arrow 2 in Fig. 3), while the second was Y_2 (arrow 3).

The aforementioned test performed to an arrangement Y_i comprises four steps. First, a new change plan C' is created, by adding the activities in Y_i to C (line 24). Second, a new set of unrefined activities A'' is built, as a result of the union of the sets A' and Y_i (line 25). This is necessary because activities in Y_i may not be refined yet, therefore requiring a future processing. Third, the impact of running activities in Y_i is computed (line 26), considering both the current view of the IT infrastructure (from I) and the changes performed so far (materialized in the snapshot s_i). The result will be stored in the snapshot s_{i+1} (in our example, s_1 represents an incremental view of the snapshot s_0, after the execution of *Install WebSrv*, *Install LibX*, and *Install WebApp*). Finally, f' is invoked recursively to refine C'', given the newly computed A'' and s_{i+1} (line 27).

Observe that the addition of the activities in Y_i to the change plan C' (line 24) takes into account dependency (pre-requisite) information. In our example, since Y_1 = {*Install WebSrv*, *Install LibX*} is a set of dependencies of *Install WebApp* (*i.e.*, *Install WebSrv* and *Install LibX* must be executed prior to *Install WebApp*), adding these activities to C'' implies in the creation of the transitions l_i = (*Install WebSrv*, *Install WebApp*) and l_{i+1} = (*Install LibX*, *Install WebApp*), and subsequent addition of l_i and l_{i+1} to the set of transitions T of the change plan C''.

Putting all the pieces together, recursive invocations of f' is the mechanism employed to navigate through all paths in the *activity dependency tree* (which represents the dependencies between software packages captured from the DML). From the example illustrated in Fig. 3, in the first invocation to f' (line 5) the activity *Install WebApp* is processed. In the first-level recursion (arrow 2 in Fig. 3) of f' (line 27), the set of immediate dependencies Y_1 is tested. Once the test fails, the recursion returns, and then the set Y_2 is tested (arrow 3). This yields a new first-level recursion (line 27). Once the test to Y_2 is successful, a second-level recursion is performed, now to process the set Y = {*Install LibY*} (arrow 4). Since *Install LibY* has no computable dependencies, a third-level recursion of f' is performed (line 20). Finally, given that there are no dependencies left to refine, the recursive refinement is finished, and the resulting refined plan C' (Fig. 3) is returned back to f' (line 14).

4.2 Snapshots of the IT Infrastructure

The concept of *snapshot* is the notion upon which the recursive search for a refined change plan is built. Having the current snapshot s_i, the refinement algorithm may foresee the new state of the IT infrastructure after the execution of the actions already computed and present in the change plan C. Consequently, it will be able to identify dependencies that are executable, and then continue the refinement process.

Fig. 4 illustrates the snapshots that are created during the refinement process of our example. In this figure, CS stands for *computer system*, OS for *operating system*, and SwElement for *software element*. The initial snapshot in our example is s_0. The two arrows from s_0 represent two possible state transitions of the IT infrastructure after the execution of each of the arrangements returned for activity *Install WebApp*. The first transition (arrow 1 in Fig. 4) leads to snapshot s_{1a}, which represents the state after the execution of (the activities in) Y_1 plus *Install WebApp*. The second transition (arrow 2), on the other hand, leads to s_{1b}, which represents the state after the execution of Y_2 (plus *Install WebApp*). The dashed arrow from s_{1a} to s_0 represents the failed test made with Y_1 (in this case, f' goes back to the previous snapshot and attempts another arrangement of immediate dependencies contained in X, Y_2). Finally, the transition from snapshot s_{1b} to s_2 (arrow 3) represents the second-level recursion to f', when the activity *Install LibY* is added to the partially refined plan C.

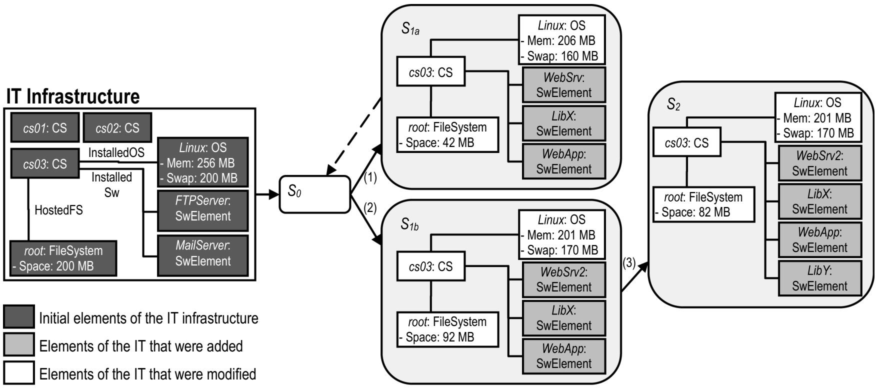

IT Infrastructure
cs01: CS
cs02: CS
cs03: CS
InstalledOS
Installed Sw
HostedFS
Linux: OS - Mem: 256 MB - Swap: 200 MB
FTPServer: SwElement
MailServer: SwElement
root: FileSystem - Space: 200 MB
Initial elements of the IT infrastructure
Elements of the IT that were added
Elements of the IT that were modified
S0
(1)
(2)
(3)
S1a
Linux: OS - Mem: 206 MB - Swap: 160 MB
cs03: CS
WebSrv: SwElement
LibX: SwElement
WebApp: SwElement
root: FileSystem - Space: 42 MB
S1b
Linux: OS - Mem: 201 MB - Swap: 170 MB
cs03: CS
WebSrv2: SwElement
LibX: SwElement
WebApp: SwElement
root: FileSystem - Space: 92 MB
S2
Linux: OS - Mem: 201 MB - Swap: 170 MB
cs03: CS
root: FileSystem - Space: 82 MB
WebSrv2: SwElement
LibX: SwElement
WebApp: SwElement
LibY: SwElement

Fig. 4. Evolution of the snapshots as the change plan is refined

Considering the representation of differences, the snapshots in Fig. 4 hold information about consumed resources and new settings present in the environment. For example, the reader may note that after the execution of activities in Y_2 and *Install WebApp*, the IT infrastructure would evolve to a new state, represented by s_{1b}. In this new state, the computer system *cs03* (*i*) has 108 MB less disk space available, and (*ii*) has the newly installed *SoftwareElements WebSrv2*, *LibX*, and *WebApp*.

Also observe that installing new software in a computer potentially increases the demand for more available physical memory (in the case of *cs03*, 55 MB more physical memory and 30 MB more swap space). Although the use of memory and swap space is flexible, the amount of such resource available for use imposes a limit, in terms of performance, in the software that may be running concurrently.

It is important to mention that the scope of the proposed snapshots is restricted to the change planning step. In addition, the information they hold is useful for the proposed refinement solution only. As a consequence, they do not take place in other phases of the change management process (*e.g.*, change testing or implementation).

4.3 Considerations on the Proposed Solution

According to the change management process, there are intermediate steps between the *design* and the actual *implementation* of a change. These steps are *assessment and evaluation*, *authorization and schedule*, and *plan updates*. The time scale to go through them may range from hours to days (or even weeks). During this period, the

IT infrastructure may evolve to a new, significantly different state (for example, due to other implemented changes). In this context, the runtime constraint-aware plan generated by our solution may not be executable upon implementation. This issue (that has been long associated with the change management process) may be tackled during the *plan updates* phase. The operator may either manually adjust the plan for the new IT scenario or re-invoke the proposed algorithm, and document the revised plan afterwards. From this point on, the time gap to implement the change should be kept to a minimum.

Another important aspect worth discussing is the *refinement flexibility* provided to the algorithm. This is regulated by the degree of detail of the preliminary plan submitted. A loosely defined preliminary plan tends to allow the algorithm to perform a broader search within the activity dependency tree. Consider, for example, an RFC to install a certain web-based application. Assuming this application depends on a Database Management System (DBMS), the operator may explicitly specify in the preliminary plan the DBMS to be installed or leave it up to the algorithm. In the latter case, the choice will be based on the alternative database packages available in the Definitive Media Library and on the runtime constraints.

To deal with the aforementioned flexibility, one could think of the existence of an *automated decision threshold*. This threshold could be specified in terms of number of software dependency levels. During the refinement process, dependencies belonging to a level above the configured threshold would be decided by the operator in an interactive fashion. Otherwise, the algorithm would do this on his/her behalf. Evaluating the pros and cons of setting a more conservative or liberal strategy is left for future work.

5 Experimental Evaluation

To prove the conceptual and technical feasibility of our proposal, we have (*i*) implemented our solution on top of the CHANGELEDGE system [5], and (*ii*) conducted an experimental evaluation considering the design and refinement of changes typically executed in IT infrastructures. Due to space constraints, we focus our analysis on five of these changes. As a result of the refinement of preliminary plans into actionable workflows, we have observed the correctness and completeness of the produced workflows (characterizing a more *qualitative* analysis of the proposed solution), in addition to performance indicators (*quantitative* analysis).

The IT infrastructure employed is equivalent to the environment of a research & development department of an organization. It is composed of 65 workstations, located in seven rooms, running either *Windows XP SP2* or *GNU/Linux*. The environment is also composed of four servers, $Server_1$, $Server_2$, $Server_3$, and $Server_4$, whose relevant settings to the context of our evaluation are presented in Table 1. Finally, the content of the Definitive Media Library is summarized in Table 2.

Table 1. Server settings

Server Name	Installed Operating System	Available Disk Space	Total Physical Memory
$Server_1$	None	20,480 MB	2,048 MB
$Server_2$	Windows 2003 Server	71,680 MB	4,096 MB
$Server_3$	Debian GNU/Linux	51,200 MB	4,096 MB
$Server_4$	Debian GNU/Linux	102,400 MB	4,096 MB

Table 2. System requirements for the software present in the DML[1]

Software Name	Disk Space	Memory	Software Dependencies
e-Commerce Web App[2]	512 MB	128 MB	SQL Server and Internet Information Server (IIS)
IIS 5.1	15 MB	16 MB	Windows XP Service Pack 2 (Win XP SP2)
IIS 7.0	15 MB	16 MB	Windows Vista Service Pack 1 (Win Vista SP1)
.Net Framework 3.5	280 MB	256 MB	Internet Explorer (IE), IIS, and Win XP SP2
SQL Server 2005	425 MB	512 MB	IE, Win XP SP2, and .Net Framework
SQL Server 2008	1,460 MB	1,024 MB	IE and Win Vista SP1
IE 7	64 MB	128 MB	Win XP SP2
Windows XP SP 2	1,800 MB	-	Windows XP
Windows Vista SP 1	5,445 MB	-	Windows Vista
Windows XP	1,500 MB	128 MB	-
Windows Vista	15,000 MB	1,024 MB	-

In regard to the submitted RFCs, the first two have as objective the installation of an e-Commerce web application (*WebApp*), one of them having *Server$_1$* as target CI and the other, *Server$_3$*. The third RFC comprises two operations: one to install and configure a network monitoring platform on *Server$_4$*, and the other to install and configure an authentication server on *Server$_3$*. The fourth RFC comprises the migration of the entire system installed on *Server$_3$* to *Server$_4$*. Finally, the fifth RFC consists in updating software packages installed in 47 out of the 65 stations that compose the IT infrastructure (typical procedure in several organizational contexts).

A partial view of the actionable workflow generated from the first RFC is presented in Fig. 5. Decision structures within the workflow were omitted for the sake of legibility. Observe that the linkage between the activities present in the workflow reflect the dependencies between the installed packages. For example, the e-Commerce Web application depends on services provided by the *SQL Server 2005* Database Management System and *Internet Information Server 5.1*. *SQL Server 2005*, in its turn, depends on the previous installation of the *.Net Framework 3.5*.

The reader may also note that implementing this actionable workflow requires, considering the information in Table 2, about 4,596 MB of disk space, and a minimum of 1,168 MB of available physical memory, from *Server$_1$*. Since this server has sufficient disk space for the installation procedures present in the workflow, the implementation of this RFC is likely to succeed. Moreover, all the installed software should execute normally, given that the target server has sufficient physical memory.

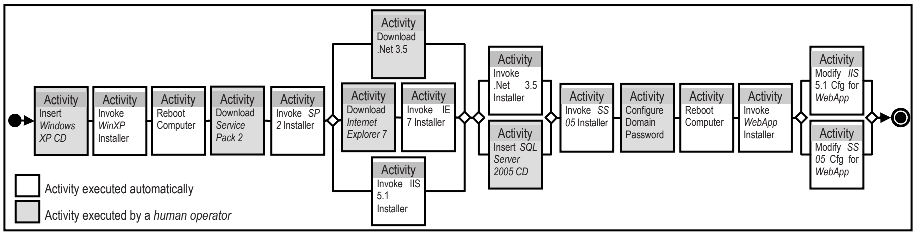

Activity Insert Windows XP CD
Activity Invoke WinXP Installer
Activity Reboot Computer
Activity Download Service Pack 2
Activity Invoke SP 2 Installer
Activity Download .Net 3.5
Activity Download Internet Explorer 7
Activity Invoke IE 7 Installer
Activity Invoke IIS 5.1 Installer
Activity Invoke .Net 3.5 Installer
Activity Insert SQL Server 2005 CD
Activity Invoke SS 05 Installer
Activity Configure Domain Password
Activity Reboot Computer
Activity Invoke WebApp Installer
Activity Modify IIS 5.1 Cfg for WebApp
Activity Modify SS 05 Cfg for WebApp
Activity executed automatically
Activity executed by a human operator

Fig. 5. Partial view of the actionable workflow for the installation of *WebApp*

[1] Source: http://www.microsoft.com

[2] The e-Commerce Web Application system requirements were estimated.

An alternative plan to the one present in Fig. 5 is the one in which *SQL Server 2008* is installed instead of *SQL Server 2005*, and *Internet Information Server 7.0*, instead of *IIS 5.1*. As a consequence, *Windows Vista* and *Windows Vista Service Pack 1* would be installed as well, instead of *Windows XP Service Pack 2* and *Windows XP*, due to the pre-requisite information. For the same reason, the installation of *.Net Framework 3.5* would not be present in this alternative plan. This plan would require 22,496 MB of available disk space from *Server*$_1$ to be executable, amount beyond the 20,480 MB currently available. Therefore, it would not be generated by our solution, since it is impractical considering the imposed resource constraints.

Table 3. Complexity of the change scenarios considering the number of activities and affected configuration items (pre and post refinement)

Scenario	Preliminary plan				Refined plan			
	Activities	Affected Stations	Affected OSes	Affected Software	Activities	Affected Stations	Affected OSes	Affected Software
1	1	1	0	1	19	1	1	6
2	1	1	0	1	23	1	1	22
3	4	2	0	2	30	2	1	26
4	46	3	0	5	182	3	1	47
5	235	47	0	6	613	47	2	29

Table 3 presents, synthetically, the computational processing spent by the CHANGELEDGE system to refine and generate actionable workflows for the five RFCs. We highlight Table 3 the number of activities, as well as the number of computer systems (stations), operating systems, and software affected in both the preliminary (specified by a human operator) and refined plans (generated by the system). Taking scenario 4 as example, one may note that the final change plan has 182 activities, automatically refined from a 40% smaller plan.

The performance of the CHANGELEDGE system to generate the actionable workflows characterized above is presented in Table 4. Our experiments were conducted on a computer equipped with a Pentium^{tm} Centrino processor, 1.7 GHz of CPU clock, 2,048 KB of cache, and 512 MB of RAM memory. The system has performed satisfactorily, demanding from a few hundreds of milliseconds (544) to a few dozens of seconds (57) to generate the aforementioned plans. We have also calculated a confidence interval of 95% for the measured times, considering 10 repetitions of the refinement process for each change document. As shown in Table 4, we expect the refinement time to vary minimally, for each scenario. The results show that our solution not only generates complete and correct plans, but has potential to reduce, in a significant way, time and efforts demanded to this end.

Table 4. Refinement processing time

Scenario	Refinement Time (ms)	Confidence Interval of the Refinement Time	
		Lower Bound (ms)	Upper Bound (ms)
1	544	535	552
2	942	937	947
3	1,754	1,736	1,771
4	3,879	3,811	3,947
5	57,674	57,482	57,866

6 Conclusions and Future Work

Change design is an undoubtedly fundamental building block of the IT change management process. However, existing computational solutions to help the generation of consistent, actionable change plans are still maturing and need more work so as to eliminate some usual simplification assumptions. In this paper, we have proposed a solution to automate the generation of change plans that take into account runtime resource constraints. This is a very important aspect to be considered in order to compute feasible plans, *i.e.*, plans in which no technical or human resource constraint is going to be violated during the execution of the plan.

The obtained results, although not exhaustive, were quite positive. The actionable workflows generated automatically from preliminary plans (designed by human operators) have respected the restrictions imposed by the target environment (*e.g.*, memory and disk space constraints). Furthermore, the refinement of change plans ran on the order of hundreds of milliseconds to dozens of seconds. This time is certainly of lower magnitude than the time that would be required by an experienced operator to accomplish the same task.

As future work we intend to investigate decision support mechanisms to help operators understand the trade-offs between alternative change designs. In addition, since our problem of IT change design concerns the realization of action sequences from a description of the goal and an initial state of the IT environment, we plan to explore how IT change design can take advantage of AI planning techniques [13]. There may be techniques from this field that our approach could benefit from, whether they are on the topic of knowledge representation, planning algorithms, or the integration of planning and scheduling.

References

1. Information Technology Infrastructure Library. Office of Government Commerce (OGC) (2008), `http://www.itil-officialsite.com`
2. IT Infrastructure Library: ITIL Service Transition, version 3. London: The Stantionery Office, p. 270 (2007)
3. Keller, A., Hellerstein, J.L., Wolf, J.L., Wu, K.-L., Krishnan, V.: The CHAMPS system: change management with planning and scheduling. In: IEEE/IFIP Network Operations and Management Symposium, vol. 1, pp. 395–408, 19–23 (2004)
4. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do internet services fail, and what can be done about it? In: 4th Usenix Symposium on Internet Technologies and Systems, Seattle, USA (2003)
5. Cordeiro, W., Machado, G., Daitx, F., et al.: A Template-based Solution to Support Knowledge Reuse in IT Change Design. In: IFIP/IEEE Network Operations and Management Symposium, Salvador, Brazil, pp. 355–362 (2008)
6. Dumitraş, T., Roşu, D., Dan, A., Narasimhan, P.: Ecotopia: An Ecological Framework for Change Management in Distributed Systems. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems IV. LNCS, vol. 4615, pp. 262–286. Springer, Heidelberg (2007)

7. Trastour, D., Rahmouni, M., Bartolini, C.: Activity-based scheduling of IT changes. In: First ACM International Conference on Adaptive Infrastructure, Network and Security, Oslo, Norway
8. Sauvé, J., Santos, R., Almeida, R., Moura, A.: On the Risk Exposure and Priority Determination of Changes in IT Service Management. In: Distributed Systems: Operations and Management, San José, CA, pp. 147–158 (2007)
9. Machado, G., Cordeiro, W., Daitx, F., et al.: Enabling Rollback Support in IT Change Management Systems. In: IFIP/IEEE Network Operations and Management Symposium, Salvador, Brazil, pp. 347–354 (2008)
10. Distributed Management Task Force: Common Information Model, `http://www.dmtf.org/standards/cim`
11. The Workflow Management Coalition Specification: Workflow Process Definition Interface - XML Process Definition Language, `http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf`
12. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, McGraw-Hill (2001) ISBN 978-0-262-53196-2
13. Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J., Wu, D., Yaman, F.: SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research 20, 379–404 (2003)

SYMIAN: A Simulation Tool for the Optimization of the IT Incident Management Process

Claudio Bartolini[1,2], Cesare Stefanelli[2], and Mauro Tortonesi[2]

[1] HP Research Labs, Palo Alto, CA, USA
claudio.bartolini@hp.com
[2] Engineering Department, University of Ferrara, Ferrara, Italy
{cstefanelli,mtortonesi}@ing.unife.it

Abstract. Incident Management is the process through which IT support organizations manage to restore normal service operation after a service disruption. The complexity of IT support organizations makes it extremely hard to understand the impact of organizational, structural and behavioral components on the performance of the currently adopted incident management strategy and, consequently, which actions could improve it. This paper presents SYMIAN, a decision support tool for the improvement of incident management performance. SYMIAN is a discrete event simulator that permits to test possible corrective measures for the IT support organization before the expensive actual implementation. SYMIAN models the IT support organization as a queuing system, considering both the time spent by operators working on incidents and the time spent when waiting for operator's availability. Experimental results show the SYMIAN effectiveness in the performance analysis and optimization of the incident resolution time for a fictitious organization designed according to real-life experiences.

Keywords: Business-driven IT management (BDIM), decision support, Information Technology Infrastructure Library (ITIL), IT service management, incident management.

1 Introduction

The IT Infrastructure Library (ITIL [1]) is a comprehensive set of concepts and techniques for managing IT infrastructure, development, and operations. Developed by the UK Office of Government Commerce (OGC), ITIL is today the de facto best practice standard for IT service management. Among the processes that ITIL defines, *Incident Management* is the process through which IT support organizations manage to restore normal service operation after a disruption, as quickly as possible and with minimum impact on the business.

Like other IT service operation processes, the incident management process has objectives that are organization-specific and defined by the business management, e.g., compliance with SLAs for some (premium) customers, minimization of economic cost in restoring service, or overall minimization of service disruption interval. The achievement of business objectives in turn requires, at the business management level, the definition and implementation of strategies in incident management.

IT support organizations need to assess their performance in dealing with service disruptions, in order to verify the effectiveness of their incident management strategies and to evaluate possible alternative strategies. Frameworks such as ITIL and COBIT [2] help by defining objectives for incident management, and usually linking them to simple high-level organization-wide performance metrics such as the mean time to incident resolution. However, the performance analysis of large IT support organizations is non-trivial and might involve a large set of complex and lower-level metrics.

The complexity of IT support organizations and the wide set of metrics to consider make it extremely hard to assess the performance of currently adopted incident management strategies. The evaluation of alternative strategies is even more difficult, as the estimation of potential improvements in incident management requires both an accurate modeling of the IT organization and the identification of critical parameters at the organizational, structural, and behavioral level on which to operate. In particular, the realignment of incident management strategies has to consider a large set of possible operations, such as restaffing (the restructuring of the support organization by increasing or cutting staffing levels, or the transfer of operators around support groups, possibly on retraining), and the implementation of different incident assignment and/or prioritization policies.

The complexity of the incident management domain makes it impossible to treat the performance optimization problem analytically, and calls for simulation-based approaches. In this context, the paper presents **SYMIAN** (SYmulation for Incident ANalysis), a decision support tool based on discrete event simulation. SYMIAN is designed to evaluate and to optimize the performance of the incident management function in IT support organizations.

SYMIAN models the IT support organization as a queuing system, an approach that is particularly well suited for the incident management application domain. In fact, it allows to distinguish the two main components of the time to resolve an incident: *working-time*, and *waiting-time*. Working-time is the time spent by operators working on trouble-tickets (*incidents* in ITIL parlance). Waiting-time is the time spent by trouble-tickets in the queues waiting for technicians to become available to operate over them or to escalate them to other parts of the organization.

SYMIAN allows users to build an accurate model of a real IT support organization and to verify its performance. In addition, SYMIAN permits to play out what-if scenarios, such as adding technicians to a given support group, merging support groups together, experimenting with alternative incident routing and/or prioritization policies, before going through the expensive and time-consuming process of implementing the actual corrective measures.

The SYMIAN tool has been applied for the performance improvement of several case studies representative of the complexity of real-life IT support organizations. The results demonstrated the effectiveness of the SYMIAN-based performance analysis and optimization process.

The paper is structured as follows. Section 2 describes the abstraction of the incident management process and the specification of the associated decision problem. Section 3 introduces the SYMIAN tool and section 4 sketches both its architecture and implementation. Section 5 presents experimental results obtained by the SYMIAN adoption in the context of a realistic case study. Section 6 reviews related work and compares our approach with it. Finally, Section 7 provides conclusive remarks and future work considerations.

2 Incident Management in IT Support Organizations

A typical IT support organization consists of a network of support groups, each comprising of a set of operators, with their work schedule. Support groups are divided into support levels (usually three to five), with lower level groups dealing with generic issues and higher level groups handling technical and time-consuming tasks. Support groups are further specialized by category of incidents that they deal with (network, server, etc…) and divided into geographies, to ensure prompt incident response (see Figure 1).

In particular, the Help Desk represents the interface for customers reporting an IT service disruption. In response to a customer request, the Help Desk "opens" an incident, sometimes called *trouble-ticket* or simply ticket. The incident is then "assigned" to a specific support group, whose technicians either fully repair the incident or "reassign" it to a different support group (usually escalating to a higher support level). As a result, an incident might have different states and be handled by different support groups throughout its lifetime. At each of these steps, the incident record is updated with the pertinent information, such as current state and related service restoration activity. If, for some reason, customers request the organization to stop working on the incident, the incident is placed in a "suspended" state to avoid incurring into SLO *(Service Level Objective)* penalties. Once the disruption is repaired, the ticket is placed in "closed" state until the end-user confirms that the service has been fully restored. In this case, the incident is "resolved" and its lifecycle ends (see Figure 2).

The complexity of IT support organizations hinders the verification of the alignment of current organizational, structural, and behavioral processes with the strategic objectives defined at the business management level. In fact, the

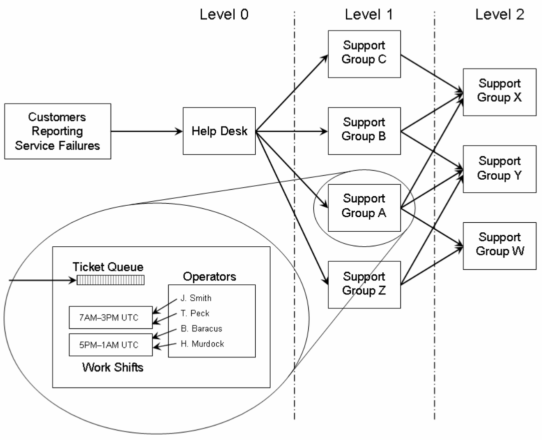


Fig. 1. Conceptual model of the IT support organization for incident management

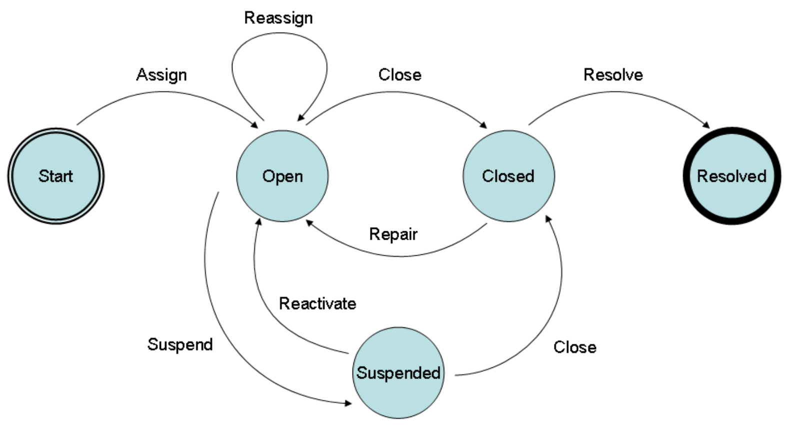


Fig. 2. Incident lifecycle

performance assessment of the incident management function is a very complex procedure which involves the business impact evaluation of the current incident management strategy, through the definition of a set of metrics that allow the objective measurement of performance indicators [3, 4]. Performance analysis and optimization are also organization-specific procedures, since the business impact of service disruptions, and consequently the metrics to consider, vary with the nature of the services and the types of disruptions that occur.

This paper does not consider the processes of business impact analysis and performance metric selection, but instead focuses on the optimization of the organizational, structural and behavioral processes for incident management according to a specified set of metrics. Hence, without loss of generality, it considers for performance optimization the ITIL-recommended objective of service disruption time minimization, and two fundamental and complementary metrics: *Mean Time To (incident) Resolution (MTTR)* and *Mean Incidents Closed Daily (MICD)*.

MTTR and MICD are organization-wide metrics, and as such they provide little insight on the internal dynamics of the organization. A comprehensive performance analysis of the incident management process has to delve into a deeper level of detail. More specifically, it needs to consider both inter- and intra- support groups dynamics, along two orthogonal dimensions: the *effectiveness* of incident routing and the *efficiency* of every single support group in dealing with the incidents. This requires taking into consideration other performance metrics which can evaluate the organization capability to directly forward incidents to the best equipped support groups and the optimality of staff allocation and operator work shift scheduling.

While the application of specific metrics for the performance evaluation of real IT support organization is almost straightforward, it is extremely difficult to evaluate the impact of changes in the organization on these metrics. As a result, the performance assessment of alternative organizations calls for decision support tools enabling *what-if scenario* analysis.

3 The SYMIAN Decision Support Tool

SYMIAN is a decision support tool for the performance analysis and optimization of the incident management function in IT support organizations. In particular, SYMIAN exploits a *discrete event simulator* to reproduce the behavior of IT organizations and to evaluate their incident management performance.

SYMIAN enables its users to play out *what-if scenarios*, allowing them to assess likely improvements in performance without having to go through the expensive and time-consuming process of implementing organizational, structural and behavioral changes. More specifically, SYMIAN allows users to incrementally specify the set of changes to apply to the current organization in order to define an alternative organization configuration that will be tested on a set of performance metrics. For instance, SYMIAN allows modifications such as re-staffing support groups, merging support groups together, experimenting with alternative work shifts, incident routing and/or prioritization policies, or other such actions. SYMIAN guides users all along the optimization process, providing ad hoc visualization of simulation results and, in case a limited set of predefined metrics such as MTTR is considered, explicit tips for the modification of some organization parameters such as the staff allocation.

SYMIAN models the IT support organization (in terms of the number of support groups, the support level, the set of operators, the operator work shifts, the relationships with other support groups, etc.) and permits to define the set of performance metrics to consider for the optimization. SYMIAN then simulates the organization behavior considering a user specified set of incidents, evaluating the desired performance metrics.

At its core, SYMIAN implements an accurate model of the IT support organization. Modeling the incident management function of IT support organizations is an arduous task. In particular, the creation of a realistic model poses two main challenges: the complexity of the IT support organization, and the extremely high volume of incidents and service calls that a typical IT support organization experiences. In addition, the effective adoption of an IT support organization in the context of a decision support tool poses significant constraints on its computational complexity. SYMIAN's model is complex enough to capture the dynamics of a real IT support organization, yet simple enough to allow for an efficient implementation and a user-friendly configuration interface.

SYMIAN models the IT support organization as a *queuing system*. More specifically, the simulated organization behavior emerges from the interaction of its support groups, which are the basic elements of the SYMIAN queuing model. In particular, each support group has a set of operators and a queue of incoming tickets. In turn, every operator has a work shift and is unavailable when off duty. When an operator is idle, he picks the ticket on top of the queue and starts working on it until the operator shift ends or the incident is resolved or cannot be further processed and needs to be forwarded to another support group. In the first case, the operator stops working on the ticket and puts it back in the incoming queue. The ticket will later be extracted from the queue following a configurable prioritization policy. Upon incident closure or escalation, the operator takes another incident from the incoming queue or remains idle if no more incidents exist.

To model the relationships between support groups, and consequently the routing of incidents through the simulated organization, SYMIAN uses a *stochastic transition matrix*. For each support group, the transition matrix describes the probability that incidents of any given category will be forwarded to a specific support group. This model builds on top of the assumption of *memory-less* incident routing, i.e., the probability of incident transition to a specific support group is independent of the history of re-assignments that the incident went through up to that moment. While this assumption allows for a considerable simplification of the model, extensive tests performed with real-life data (using the same dataset as in [5]) on the SYMIAN tool demonstrated that the model behaves with excellent fidelity. A full discussion of the SYMIAN model validation is beyond the scope of the present paper.

Incidents are injected into the system by an incident generation entity which models the aggregate behavior of customer incident reports. An accurate model of the incident arrival/generation process is of critical importance for a realistic simulation. To ensure a realistic input for the simulation, one possibility is to use traces of incidents obtained from the analysis of the operational logs in real IT support organizations. However, considering only real incident traces would limit the applicability of the simulative approach to a small set of predefined input, thus preventing its use to verify how the modeled organization would behave under heavy incident load or under a specific set of incidents following a given inter-arrival or severity pattern. As a result, there is the need to consider synthetic incident generation according to configurable stochastic patterns.

To this end, SYMIAN allows for a highly configurable stochastic incident generation. More specifically, SYMIAN divides incidents in several categories, according to the amount of work they require for service restoration at every support level. In addition, every incident category has several levels of severity, with an increasing (average) time to incident closure or escalation to a higher level support group. Every specific category and severity couple is assigned a random probability distribution which allows the configuration of the amount of work required by incidents. Incident inter-arrival time is also stochastically modeled according to a random variable distribution.

4 SYMIAN: Architecture and Implementation

The architecture of the SYMIAN tool is depicted in Figure 3, that shows its main components: the *User Interface* (UI), the *Configuration Manager* (CM), the *Simulator Core* (SC), the *Data Collector* (DC), and the *Trace Analyzer* (TA).

The User Interface component allows users to load simulation parameters from a file, to change current simulation parameters, to save current simulation parameters to file, and to start simulations. UI provides both an interactive textual and a non-interactive command-line interface.

The Configuration Manager takes care of the simulator configuration, enforcing the user-specified behaviors, e.g., with regards to verbosity of tracing information, and simulator parameters, e.g., the characterization of incident generation, the number

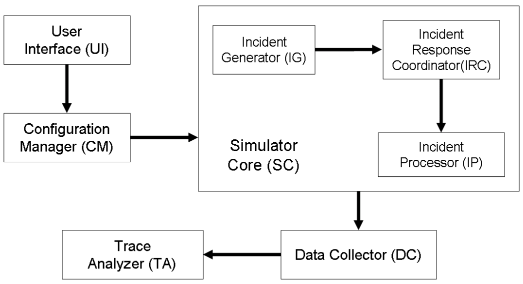


Fig. 3. Architecture of the SYMIAN tool

and size of support groups, and the relationships between support groups, in the domain specific model recreated by the Simulator Core component.

The Simulator Core component implements the domain specific model. SC has three sub-components: *Incident Generator* (IG), *Incident Response Coordinator* (IRC) and *Incident Processor* (IP). The Incident Generator generates incidents according to a random distribution pattern which follows user-specified parameters, and injects them into the system. The Incident Response Coordinator receives incidents and dispatches them to the processing domain entities (support groups), which are in turn implemented by the Incident Processor.

The Data Collector component collects data from the simulation that can be post-processed to assess the performance of incident management in the modeled organization. In particular, DC performs an accurate monitoring of support group status, in terms of incoming incident queue size and operator activity, and a careful tracking of incidents status. DC saves its simulation results data in a file that users can then analyze with the Trace Analyzer component.

SYMIAN is implemented in the Ruby (*http://www.ruby-lang.org/*) programming language. Ruby was chosen for its remarkable extensibility and its support for meta-programming. The capability to easily redefine the behavior of time-handling classes in the Ruby standard library allowed the implementation of a simulated clock which models the flow of simulation-time in a very similar way to what happens in real life. In addition, Ruby's meta-programming enabled the definition of domain-specific languages and their use in the realization of several simulator components. These have proved to be particularly effective development techniques.

The availability of a wide range of high-quality scientific libraries was also a major reason behind the adoption of Ruby. In particular, SYMIAN exploits the GNU Scientific Library (GSL), via the Ruby/GSL bindings, for high-quality random number generation, and it integrates with the Gnuplot data visualization tool to plot some of the simulation results. Finally, SYMIAN exploits Ruby facilities to import configuration parameters and export simulation results in the XML, YAML, and CSV formats, in order to ease integration with external software for the automation of multiple simulation runs and with scientific tools for post processing of simulation results.

5 Experimental Results

This section presents an experimental evaluation of the SYMIAN effectiveness in the performance analysis and optimization of the incident management process. More specifically, SYMIAN is applied to minimize the service disruption time in the context of a case study IT support organization, with the constraint of preserving the current number of operators.

As a result, the objectives of the performance improvement process are the maximization of the mean incidents closed daily (*MICD*) metric, as well as the minimization of the mean time to resolution (*MTTR*) metric.

The target of this experimental evaluation is the fictitious incident management organization *INCS'R'US*, which is composed of 3 support levels (0-2), 31 support groups, and 348 operators. The complete characterization of the 31 support groups is presented in Table 1. To limit the complexity of the case study, the routing of incidents in the INCS'R'US organization is assumed to be unidirectional, that is support groups of level N can only receive incidents from support groups of level N-1 and escalate incidents to support groups of level N+1. In addition, an equal probability of incident escalation to each of the support groups of immediately higher level is assumed.

INCS'R'US deals with incidents modeled according to the characterization provided in Table 2. Incidents have 4 categories (A-D) and 3 severity levels (1-3). For every specific combination of incident category and severity, the amount of work that incidents require for service restoration, at every support level, follows a uniform random probability distribution. In Table 2, the abbreviated notation U(α), where α > 0, represents the uniform random variable distribution in the [0, α] interval.

Table 1. Support group characterization in the *Incs'R'Us* incident management organization

Support Level	Support Group (Number of Operators)	Work Shift
0	Help Desk (75)	(25 operators) 7AM-3PM UTC
		(25 operators) 4AM-12PM UTC
		(25 operators) 12PM-8PM UTC
		(10 operators) 5PM-1AM UTC
1	SG1 (15), SG9 (12), SG15 (13), SG18 (5)	7AM-3PM UTC
	SG2 (7), SG10 (7), SG13 (7)	8AM-4PM UTC
	SG3 (15), SG19 (12)	12PM-8PM UTC
	SG4 (4), SG11 (6)	2PM-10PM UTC
	SG5 (14), SG16 (12), SG20 (6)	4AM-12PM UTC
	SG6(12), SG17 (9)	3AM-11AM UTC
	SG7 (5), SG14 (5)	5PM-1AM UTC
	SG8 (6), SG12 (8)	9AM-5PM UTC
2	SG21 (9), SG25 (10)	2PM-10PM UTC
	SG22 (8), SG26 (8)	9AM-5PM UTC
	SG23 (7), SG27 (7)	8AM-4PM UTC
	SG24 (9), SG28 (10)	5PM-1AM UTC
	SG29 (9)	3AM-11AM UTC
	SG30 (6)	4AM-12PM UTC

Table 2. Stochastic characterization of the amount of work time (in seconds) required for incident closure

	Severity Level 1	Severity Level 2	Severity Level 3
Category A	L0: U(300) L1: 0 L2: 0	L0: U(900) L1: U(240) L2: 0	L0: U(1800) L1: U(900) L2: U(120)
Category B	L0: U(300) L1: U(1200) L2: U(120)	L0: U(600) L1: U(2400) L2: U(240)	L0: U(900) L1: U(3600) L2: U(480)
Category C	L0: U(600) L1: U(150) L2: U(1200)	L0: U(900) L1: U(300) L2: U(2400)	L0: U(1200) L1: U(450) L2: U(3600)
Category D	L0: U(900) L1: U(1200) L2: U(1200)	L0: U(1800) L1: U(4800) L2: U(4800)	L0: U(2400) L1: U(6000) L2: U(6000)

Category A models incidents which mostly require work at support level 0, and a limited amount of work at higher support levels. Category B and C model incidents which require work at every support level, but mostly at support level 1 and 2 respectively. Category D models incidents which require a significant amount of work at every support level. For every incident, category and severity level are randomly chosen, with uniform probability, at generation time. Incident inter-arrival times follow a random exponential probability distribution with an average of 30 seconds.

A first simulation was conducted to evaluate the performance of the current organization. The simulation covered three whole days of simulated time, starting from 2PM UTC. The first 24 hours of simulated time were not considered for the evaluation of the performance metrics, and were introduced only to prime the simulation environment to avoid taking measurements on a cold start. Table 3 (first column) provides the values for the MICD and MTTR performance metrics obtained from the simulation. The table also shows the Mean Work Time (MWT) metric, defined as the mean work time per closed incident, as an indication on the amount of work spent on service restoration.

By analyzing the variation of the incident queue size at every support group using both SYMIAN graphical visualization and time series analysis functions, it was easy to realize that support groups SG1, SG4, SG7, SG8 and SG14 at support level 1 and support group SG30 at support level 2 were a major performance bottleneck, while the Help Desk and support groups SG3 and SG17 were oversized. As an example of the effectiveness of visual analysis to locate performance bottlenecks, Figure 4 (a) plots the variation of incident queue size at support group SG30.

To improve the organization performance, 8 operators were transferred from the Help Desk to support groups SG1, SG4, SG7, and SG8 (2 operators for each group), 3 operators were transferred from support group SG3 to support group SG14, and 2 operators were transferred from support group SG17 to support group SG30. A new simulation was then launched to assess the performance of the new organization. Table 3 (second column) and Figure 4 (b) provide respectively the performance metrics and the variation of incident queue size at support group SG30 for the new simulation.

Table 3. Performance metrics from the first and second simulation

	First simulation	Second simulation
Total incidents generated	8609	8609
Incidents generated after warm-up	5728	5728
MICD	1811	2002
MTTR (in seconds)	53423	47047
MWT (in seconds)	L0: 508, L1: 809, L2: 784	L0: 506, L1: 811, L2: 773

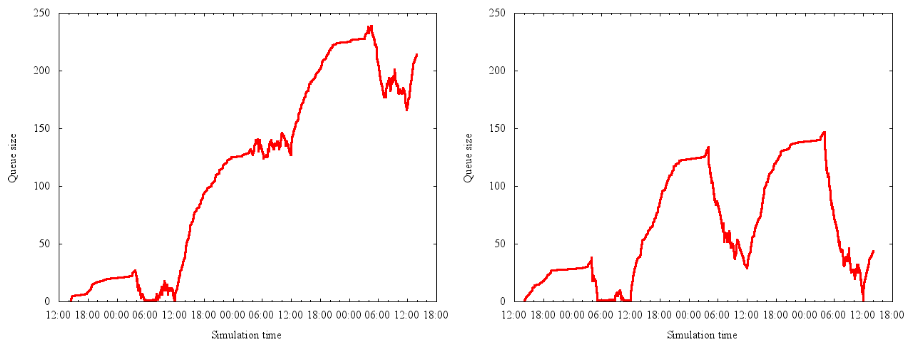


Fig. 4. Incident queue size at support group SG30 during the first (a) and second (b) simulation

The results of the second simulation proved that the reallocation of operators was very effective in improving the whole system performance. In particular, the `INCS'R'US` organization exhibited a 10.5% improvement of the MICD and a 11.9% decrease of the MTTR.

Although the target of the previous performance optimization experiment is a fictitious organization, the case study was carefully designed to be representative of the complexity of real-life IT organizations. Therefore, the simulation results demonstrate the effectiveness of the `SYMIAN` tool for the performance optimization of the incident management function in IT support organizations.

6 Related Work

The present work contributes to the up and coming research domain of Business-driven IT management (BDIM), which builds on the tradition of the research in network, system and service management. BDIM has been defined as "*the application of a set of models, practices, techniques and tools to map and to quantitatively evaluate interdependencies between business performance and IT solutions – and using the quantified evaluation – to improve the IT solutions' quality of service and related business results*". For a thorough review of BDIM, see [6].

Some notable early works in BDIM include applications to change management [7, 8, 9], capacity management [10, 11, 12], network security [13], and network configuration management [14]. All these research efforts (possibly with the exception of [14]), limit their scope to the *technology* dimension of IT management, thereby focusing on the fine tuning of systems configuration and on the introduction of automation as means to improve the IT management processes.

The present work, instead, belongs to a recently emerged research area that focuses on the other two fundamental dimensions of IT management: *people* and *processes*. The interest on this topic arose as researchers started analyzing the relationships between people, processes and technological optimization and the impact of automation and process complexity on labor cost. As a representative example, we cite Diao et al.'s recent research effort addressing the very important question of when does it make sense to automate processes based on metrics of process complexity [15, 16]. The main difference between our approach and theirs is that our focus is in achieving significant improvements in the performance of the organization through decision support and simulation techniques. In this context, in previous works we have extensively studied the business impact of incident management strategies [3, 4], using a methodology that moved from the definition of business-level objectives such as those commonly used in balanced scorecards [17]. With respect to those works, this paper follows a novel approach that the first time proposes and implements detailed modeling of the inner functioning of the IT support organization to support what-if scenario analyses.

The analysis of the incident management process and the IT support organization model that we present in this paper share is founded on our work presented in [5]. However, here we push our modeling effort far beyond the definition of metrics for the performance assessment of IT support organizations that we conducted in [5], all the way to the design and implementation of the SYMIAN decision support tool.

7 Conclusions and Future Work

The performance optimization of large-scale IT support organization can be extremely complex and might require additional help from decision support tools. This paper presented the SYMIAN tool for the performance optimization of incident management in IT support organizations. The application of SYMIAN in case studies expressively designed to capture the complexity of real-life IT support organizations demonstrated the tool effectiveness in the difficult performance analysis and improvement process.

Future versions of SYMIAN will be complemented with the application of automated techniques for the optimization of parameters, e.g., staff allocation, in the context of specified performance metrics. The IT support organization model implemented in SYMIAN is also currently being extended to consider operators with *skills* that skew their expected working time for incidents of a given category and priority policies in extracting incidents from queues.

Finally, a more comprehensive version of the SYMIAN tool will link performance optimization metrics with key performance indicators or impact metrics that are meaningful at the business level.

References

[1] IT Infrastructure Library, ITIL Service Delivery and ITIL Service Support, OGC, UK (2003)

[2] IT Governance Institute, COBIT 3rd edns (2000), http://www.isaca.org/COBIT.htm

[3] Bartolini, C., Sallé, M.: Business Driven Prioritization of Service Incidents. In: Proceedings of Distributed Systems Operations and Management (DSOM) (2004)

[4] Bartolini, C., Sallé, M., Trastour, D.: IT Service Management driven by Business Objectives – An Application to Incident Management. In: Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2006) (April 2006)

[5] Barash, G., Bartolini, C., Wu, L.: Measuring and Improving the Performance of an IT Support Organization in Managing Service Incidents. In: Proc. 2nd IEEE Workshop on Business-driven IT Management (BDIM 2007), Munich, Germany (2007)

[6] Moura, A., Sauvé, J., Bartolini, C.: Research Challenges of Business–Driven IT Management. In: Proceedings of the 2nd IEEE / IFIP International Workshop On Business-Driven IT Management (BDIM 2007), Munich, Germany (2007)

[7] Keller, A., Hellerstein, J., Wolf, J.L., Wu, K., Krishnan, V.: The CHAMPS System: Change Management with Planning and Scheduling. In: Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2004). IEEE Press, Los Alamitos (2004)

[8] Sauvé, J., Rebouças, R., Moura, A., Bartolini, C., Boulmakoul, A., Trastour, D.: Business-driven decision support for change management: planning and scheduling of changes. In: State, R., van der Meer, S., O'Sullivan, D., Pfeifer, T. (eds.) DSOM 2006. LNCS, vol. 4269. Springer, Heidelberg (2006)

[9] Trastour, D., Rahmouni, M., Bartolini, C.: Activity-Based Scheduling of IT Changes. In: Bandara, A.K., Burgess, M. (eds.) AIMS 2007. LNCS, vol. 4543, pp. 73–84. Springer, Heidelberg (2007)

[10] Aiber, S., Gilat, D., Landau, A., Razinkov, N., Sela, A., Wasserkrug, S.: Autonomic Self–Optimization According to Business Objectives. In: Proceedings of the International Conference on Autonomic Computing (2004)

[11] Menascé, D., Almeida, V.A.F., Fonseca, R., Mendes, M.A.: Business-Oriented Resource Management Policies for e-Commerce Servers. In: Performance Evaluation, vol. 42, pp. 223–239. Elsevier Science, Amsterdam (2000)

[12] Sauvé, J., Marques, F., Moura, A., Sampaio, M., Jornada, J., Radziuk, E.: SLA Design from a Business Perspective. In: Schönwälder, J., Serrat, J. (eds.) DSOM 2005. LNCS, vol. 3775. Springer, Heidelberg (2005)

[13] Wei, H., Frinke, D., Carter, O., et al.: Cost–Benefit Analysis for Network Intrusion Detection Systems. In: Proceedings of the 28th Annual Computer Security Conference (October 2001)

[14] Boutaba, R., Xiao, J., Aib, I.: CyberPlanner: A Comprehensive Toolkit for Network Service Providers. In Proceedings of the 11th IEEE/IFIP Network Operation and Management Symposium (NOMS 2008), Salvador de Bahia, Brazil (2008)

[15] Diao, Y., Keller, A., Parekh, S., Marinov, V.: Predicting Labor Cost through IT Management Complexity Metrics. In: Proceedings of the 10th IEEE/IFIP Symposium on Integrated Management (IM 2007), Munich, Germany (2007)

[16] Diao, Y., Bhattacharya, K.: Estimating Business Value of IT Services through Process Complexity Analysis. In: Proceedings of the 11th IEEE/IFIP Network Operation and Management Symposium (NOMS 2008), Salvador de Bahia, Brazil (2008)

[17] Kaplan, R., Norton, D.: The Balanced Scorecard: Measures that Drive Performance. Harvard Business Review 70(1), 71–79 (1992)

Flexible Resolution of Authorisation Conflicts in Distributed Systems[*]

Changyu Dong, Giovanni Russello, and Naranker Dulay

Department of Computing, Imperial College London
180 Queen's Gate, London, SW7 2AZ, UK
{changyu.dong,g.russello,n.dulay}@imperial.ac.uk

Abstract. Managing security in distributed systems requires flexible and expressive authorisation models with support for conflict resolution. Models need to be hierarchical but also non-monotonic supporting both positive and negative authorisations. In this paper, we present an approach to resolve the authorisation conflicts that inevitably occur in such models, with administrator specified conflict resolution strategies (rules). Strategies can be global or applied to specific parts of a system and dynamically loaded for different applications. We use Courteous Logic Programs (CLP) for the specification and enforcement of strategies. Authorisation policies are translated into labelled rules in CLP and prioritised. The prioritisation is regulated by simple override rules specified or selected by administrators. We demonstrate the capabilities of the approach by expressing the conflict resolution strategy for a moderately complex authorisation model that organises subjects and objects hierarchically.

1 Introduction

In modern enterprise systems restricting access to sensitive data is a critical requirement. However, this usually involves a large number of objects which may have different security requirements. The complexity of managing such systems results in high administrative costs and long deployment cycles. This worsens as a system expands because the effort and time required for management becomes a burden. It is important that security management procedures are simplified and automated to reduce administrative costs [1,2]. Policy-based management is potentially a more appropriate solution where security management includes support for the specification of authorisation policies, and the translation of these policies into information which can be used by enforcement mechanisms.

[*] This research was supported by the UK's EPSRC research grant EP/C537181/1 and forms part of CareGrid. We also acknowledge financial support in part from the EC IST EMANICS Network of Excellence (26854). The authors would like to thank the members of the Policy Research Group at Imperial College for their support and to Jorge Lobo at IBM T. J. Watson research center for his advice.

Early authorisation models were usually monotonic and supported only one type of policy. Most recent authorisation models are non-monotonic and support both positive and negative authorisation policies [3,4,5,6,7]. The advantage of supporting both is greater flexibility, expressiveness and convenience. However, conflicts can arise when both positive and negative policies are applicable at the same time. This type of conflict is referred to as a *modality conflict* [8]. When applying a non-monotonic authorisation model in large distributed systems, modality conflicts are hard to avoid. They can arise due to many reasons, such as omissions, errors or conflicting requirements. Therefore, conflict resolution is an important practical requirement in systems that support non-monotonic authorisation.

Many conflict resolution strategies have been developed. The most primitive conflict resolution rules for authorisation policies include:

– Negative (positive) takes precedence. If a conflict arises, the result will be negative (positive).
– Most specific (general) takes precedence. When policies defined at different levels in a hierarchy conflict, the most specific (general) one wins.
– Strong and weak. Some policies are marked as strong authorisations and others are marked as weak. In case they conflict, strong policies win.

However, real application may become so complex that using only one rule may not guarantee solving the conflict (for example, when two strong policies conflict). In these cases, either the conflict ends up undecided or further resolution rules must be applied to resolve the conflict. As a consequence, more and more sophisticated conflict resolution rules are designed to fulfill the requirements posed by applications.

On the other hand, different applications usually have different need for conflict resolution. For example, negative takes precedence would be one requirement in military systems while positive takes precedence may be preferred in open systems. To make an authorisation model flexible, it is better not to fix the conflict resolution strategy at design time, but leave it to the system administrators to decide what is the appropriate strategy for a system or sub-system. This raises the questions: how to express the variety of conflict resolution requirements that exist in real-world applications and how to make it easy for administrators to express such requirements. In this paper, we show how to capture and define authorisation conflict resolution strategies using small Courteous Logic Programs (CLP) [9].

We chose CLP for three main reasons: (1) it has a clear well-defined semantics that is easy to understand; (2) the conflict handling in CLP is context-neutral, i.e. it does not depend on what the application is, or on what hierarchies are being used. This feature is vital to many policy frameworks, which must be flexible in order to serve different applications; (3) the declarative nature of logic programs makes it possible to separate the conflict resolution rules from the implementation details, and therefore makes it possible to define resolution rules as reusable meta-policies that can be published and used by different organisations with only minimal changes to their existing policy management system.

2 Courteous Logic Programs

2.1 Overview

Courteous Logic Programs (CLP) is motivated largely by extended logic programs [10] where classical negation is permitted in rule heads and bodies. Extended logic programs are more expressive than normal logic programs where the negative information is implicit. For example the statement: "Penguins do not fly" can be expressed naturally as:

$$\neg fly(X) \leftarrow penguin(X).$$

However, explicit negation may also lead to contradictions. For example, with the above statement and the following ones:

$$\begin{aligned} fly(X) &\leftarrow bird(X). \\ bird(X) &\leftarrow penguin(X). \\ &penguin(tweety). \end{aligned}$$

We can derive both $\neg fly(tweety)$ and $fly(tweety)$, which contradict each other. The design goal of CLP is to preserve the expressive power brought by explicit negation in extended logic programs while also guaranteeing a consistent and unique set of conclusions. This is done by labelling the logic rules and prioritising the labels. If two conflicting conclusions can be derived, the conclusion being derived from a rule with a higher priority label will override the one with a lower priority label. In the case that each conclusion is derived from multiple rules with different priorities, the conclusion from the rule with the highest priority wins.

We illustrate the intuition with a simple example. If we label the rule "birds fly" with a label $\langle bird\rangle$, and the rule "Penguins do not fly" with a label $\langle penguin\rangle$. Furthermore is we assume that there exists a super-penguin which can fly, and its name is Tweety, then we have the following additional rules:

$$\begin{aligned} \langle superPenguin\rangle\ fly(X) &\leftarrow superPenguin(X). \\ &superPenguin(tweety). \end{aligned}$$

Of course by knowing X is a penguin, we can draw a more precise conclusion about X than only knowing X is a bird. We would decide that $\langle penguin\rangle$ has a higher priority than $\langle bird\rangle$, and $\langle superPenguin\rangle$ has a higher priority than $\langle penguin\rangle$.

The conclusion $fly(tweety)$ can be drawn by two rules with labels $\{\langle bird\rangle, \langle superPenguin\rangle\}$ and $\neg fly(tweety)$ can be drawn by one rule with label $\{\langle penguin\rangle\}$. Although $\langle penguin\rangle$ overrides $\langle bird\rangle$, it is overriden by $\langle superPenguin\rangle$. Therefore the conclusion $fly(tweety)$ is the winner because it is supported by the rule with the highest priority.

CLP is also computationally tractable for the (acyclic) propositional case, e.g. under the Datalog restriction. The entire answer set can be computed in

$O(m^2)$time, where m is the size of the ground-instantiated program. This makes CLP more attractive especially in our case because most of the authorisation policies can be expressed in Datalog.

2.2 Syntax

A courteous logic program can be viewed as a union of two disjoint parts: the *main sub-program* and the *overrides sub-program*.

The main sub-program is defined as labelled rules. A labelled rule has the form:

$$\langle lab \rangle \; L_0 \leftarrow L_1 \wedge \ldots \wedge L_m \wedge \sim L_{m+1} \wedge \ldots \wedge \sim L_n$$

where *lab* is an optional label for the rule, each L_i is a literal. If a rule has a label, the label is preserved during instantiation, all the ground instances of the rule have the same label. A literal can be of the form A or $\neg A$ where A is an atom and $\neg$ is the classical negation operator. $\sim$ stands for negation-as-failure.

A special binary predicate *overrides* is used to specify prioritisation. *overrides* (i, j) means that the label i has strictly higher priority than the label j. *overrides* is syntactically reserved and cannot appear in the rule body. Given a set *Lab* of all the labels in the program, *overrides* must be a strict partial order over *Lab*, i.e. transitive, antisymmetric and irreflexive.

The program must be acyclic and stratified which means one can restructure the program into separate parts in such a way that references from one part refer only to previously defined parts.

2.3 Semantics

The semantics of CLP is defined using the concept of an *answer set*. Let $\mathcal{C}$ be a courteous logic program, it has a unique answer set S which is defined as follows.

We use $\mathcal{C}^{instd}$ to denote the logic program that results from each rule in $\mathcal{C}$ having variables been replaced by the set of all its possible ground instantiations. Let$\rho = p_1, ..., p_m$ be a sequence of all the ground atoms of $\mathcal{C}^{instd}$ such that ρ is a reverse-direction topological sort of the atom dependency graph. Reverse means that body comes before head. We call ρ a total atom stratification of $\mathcal{C}$. For example, given the following logic program:

$$p(a) \leftarrow q(a).$$
$$p(a) \leftarrow p(b).$$

A total atom stratification is $q(a), p(b), p(a)$. There may be multiple total stratifications, but the answer set is independent of the total stratification choice.

Given a ρ such that all of the overrides atoms come before all the other atoms and let p_i be the i^{th} ground atom in ρ, the answer set is constructed iteratively:

$$S_0 = \emptyset$$
$$S_i = \bigcup_{j=1,\ldots,i} T_j, i \geq 1$$

$$S = \bigcup_i T_i$$

where $\emptyset$ is the empty set and T_i is defined as follows:

$$T_i = \{\sigma p_i | Cand_i^{\sigma} \neq \emptyset, \forall k \in Cand_i^{\neg\sigma}.\exists j \in Cand_i^{\sigma}.overrides(j,k) \in S_{i-1}\},$$
$$Cand_i^{\sigma} = \{j | labels(j,r), Head(r) = \sigma p_i, Body(r) \subseteq S_{i-1}\}$$

Here σ stands for a modality, either positive (+, usually omitted) or negative ($\neg$). $\neg\sigma$ means the reverse modality of σ. $label(j,r)$ means j is the label for rule r. $Head(r)$ is a ground literal in the head of the rule r, $Body(r)$ is the set of all ground literals in the body of r. The set T_i either consists of a single grounded literal, or is empty. The literal is of positive sign (p_i), or of negative sign ($\neg p_i$).

According to above definition, $\mathcal{C} \models p$ means that p is in the answer set of $\mathcal{C}$.

3 Overview

Our approach is depicted in Figure 1. When a CLP-enabled policy system is asked to decide on whether a $(subject, target, action)$ request should be permitted, it translates the authorisation policies associated with the request from the underlying authorisation model (e.g. XACML) into labelled rules in CLP (the main sub-program). The labelled rules for the request are then forwarded to the policy decision point that makes the authorisation and uses the conflicts resolution rules (the overrides sub-program) to resolve any conflicts by choosing the rules with the highest priorities. If a system supports multiple authorisation models, then a translation module is needed for each. Systems can also select the conflict resolution rules needed on an application-by-application basis.

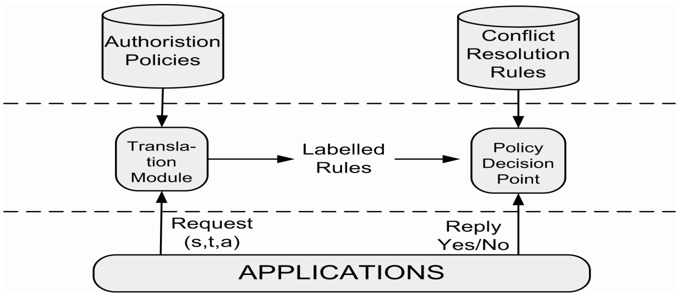


Fig. 1. Architecture of CLP-enabled policy system

4 Case Study: Hierarchical Conflict Resolution

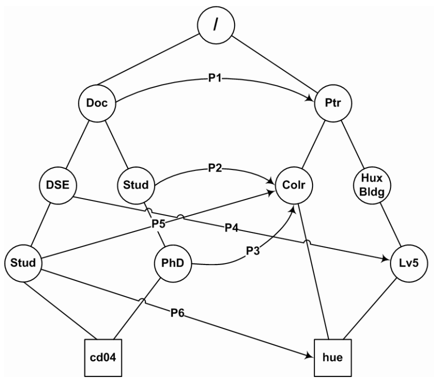
To demonstrate the capabilities of CLP-based conflict resolution, we apply it to a moderately complex authorisation model, Ponder2 [11]. Ponder2 supports hierarchies that are used to define objects, entities and organisational structures at different levels of scale from small embedded devices to complex services and virtual organisations. It supports both positive and negative authorisation policies. In Ponder2, managed objects (MOs) are organised in a tree-like domain hierarchy according to various criteria such as geographical boundaries, object type, responsibility and authority. Each domain in the hierarchy contains other domains or MOs. Instances of MOs are allowed to be present in more than one domain. In this way, if a domain represents a role then an instance of a MO can be associated with multiple roles. The rationale for the authorisation model and conflict resolution strategy are described in more detail in [12].

4.1 Domain Hierarchy and Authorisation Policies

Figure 2 shows a small concrete example of a hierarchically organised set of Ponder2 authorisation policies for printers in a department. In the figure, each circle represents a domain and the squares are MOs. A domain or an MO can be addressed by its domain path which is a Unix-style path from the root domain to the specified domain or MO. An MO may have multiple domain paths since it can be in more than one domain. For example, there are two domain paths for *cd*04: */Doc/DSE/Stud* and */Doc/Stud/PhD*.

Authorisation policies define what actions a subject can(not) invoke on a target. If a domain is used as a subject or as a target, then the member MOs of the domain inherit the policies defined for the domain. For example, in the domain hierarchy shown in Figure 2, an authorisation policy *P*1 exists with subject "*/Doc*", target "*/Ptr*" and action "print", then when MO *cd*04 sends a request for action print to MO *hue*, the policy must be considered in the authorisation process.

The syntax of an authorisation policy is simple. **auth+** means this policy is a positive authorisation policy and **auth-** means it is negative. **final** is an optional keyword which gives authorisation policies more priority than non-final



P1: auth+ (*/Doc*).*print →/Ptr*
P2: auth- (*/Doc/Stud*).*print→ /Ptr/Colr*
P3: auth+ (*/Doc/Stud/PhD*).*print →/Ptr/Colr*
P4: final auth+ (*/Doc/DSE*).*print → /Ptr/HuxBldg/Lv5*
P5: auth- (*/Doc/DSE/Stud*).*print → /Ptr/Colr*
P6: auth+ (*/Doc/DSE/Stud*).*print → /Ptr/Colr/hue*

Fig. 2. Domains and Authorisation Policies for Printers in a Department

ones. Constraints can be used to limit the applicability of policies. In the example shown in Figure 2, we can see there are six policies defined at different levels between different domains. We summarise the policies below.

$P1$ is a general policy which gives all the members in the Department of Computing (Doc) access to all the printers (Ptr). Policy $P2$ is a negative policy which prevents students (Stud) from using colour printers (Colr). However, $P3$ says that if a student is registered as a PhD student (PhD), then he can use colour printers. $P4$ is a final policy which permits all the members of the distributed software engineering (DSE) group to use all the printers located at level 5 of the Huxley Building. $P5$, $P6$ together say that the students in the DSE group cannot access colour printers other than hue.

4.2 Translating Authorisation Policies to CLP Rules

When cd04 sends a print job to hue, the policies in Figure 2 are translated into the labelled rules shown in Figure 3. The translation algorithm is specific to the authorisation model. The translation module for Ponder2 evaluates all the possible domain paths. The domain paths for cd04 are: $\{/Doc/DSE/Stud/cd04$, $/Doc/Stud/PhD/cd04\}$. There are two domain paths for hue: $\{/Ptr/Colr/hue$, $/Ptr/HuxBldg/Lv5/hue\}$. For simplicity, we refer to these paths as p_{s1}, p_{s2}, p_{t1}, p_{t2} afterwards. For each combination (p_{si}, p_{tj}), two rules are added: one with a label $\langle (p) \rangle$ and the other with a label $\langle (n) \rangle$. The first 8 rules in Figure 3 apply the principle that the overall authorisation is the aggregation of the path authorisation (see section 4.3). The translation module also generates a rule for the default authorisation policy labelled $\langle (d) \rangle$.

The remaining 9 labelled rules in Figure 3 are generated for the actual policies applied along a specific path combination. For each policy, the translation module translates it into a labelled rule with head $auth(p_{si}, p_{tj}, action)$ if the policy is $auth+$, and $\neg auth(p_{si}, p_{tj}, action)$ if the policy is $auth-$. These labelled rules are used to derive the path authorisation for each path combination. The label is of the form $(type, tdis, sdis, mode)$. $type$ is determined by the policy type. If the policy is a final policy, then $type = f$. Otherwise $type = n$, which means it is a normal policy. $tdis, sdis$ are the distances regarding the path combination (p_{si}, p_{tj}). If we view the domain hierarchy as a graph and each policy defined as an arc between two nodes, then $tdis$ is the number of nodes traversed from the subject to the target through the "policy arc" and $sdis$ is the number of nodes traversed from the subject to the node where the arc starts. Finally, $mode$ is the modality of the policy. For an $auth+$ policy, $mode = p$, for an $auth-$ policy, $mode = n$.

Let's take the (p_{s1}, p_{t1}) combination as an example to show how the policies are mapped into labelled rules. The policies relevant to this combination are $P1, P5, P6$. $P1$ is a positive authorisation policy, so it is mapped into a rule with head $auth(/doc/dse/stud/cd04, /ptr/colr/hue, print)$. The label of this rule is $\langle (n, 5, 3, p) \rangle$. Recall that $P1$ is a normal positive policy defined over $(/doc, /ptr, print)$. In the label, n means a normal policy, p means positive. The distance from cd04 to the domain $/doc$ is 3, the distance from hue to the

%$auth(cd04, hue, print)$ defined for each path combination.
$\langle(p)\rangle\ auth(cd04, hue, print) \leftarrow auth(/doc/dse/stud/cd04, /ptr/colr/hue, print).$
$\langle(n)\rangle\ \neg auth(cd04, hue, print) \leftarrow \neg auth(/doc/dse/stud/cd04, /ptr/colr/hue, print).$

$\langle(p)\rangle\ auth(cd04, hue, print) \leftarrow auth(/doc/dse/stud/cd04, /ptr/huxBldg/lv5/hue).$
$\langle(n)\rangle\ \neg auth(cd04, hue, print) \leftarrow \neg auth(/doc/dse/stud/cd04, /ptr/huxBldg/lv5/hue).$

$\langle(p)\rangle\ auth(cd04, hue, print) \leftarrow auth(/doc/stud/phd/cd04, /ptr/colr/hue, print).$
$\langle(n)\rangle\ \neg auth(cd04, hue, print) \leftarrow \neg auth(/doc/stud/phd/cd04, /ptr/colr/hue, print).$

$\langle(p)\rangle\ auth(cd04, hue, print) \leftarrow auth(/doc/stud/phd/cd04, /ptr/huxBldg/lv5/hue).$
$\langle(n)\rangle\ \neg auth(cd04, hue, print) \leftarrow \neg auth(/doc/stud/phd/cd04, /ptr/huxBldg/lv5/hue).$

%default authorisation
$\langle(d)\rangle\ \neg auth(cd04, hue, print).$

%the first path combination,policies P1, P5 and P6
$\langle(n, 5, 3, p)\rangle\ auth(/doc/dse/stud/cd04, /ptr/colr/hue, print).$
$\langle(n, 2, 1, n)\rangle\ \neg auth(/doc/dse/stud/cd04, /ptr/colr/hue, print).$
$\langle(n, 1, 1, p)\rangle\ auth(/doc/dse/stud/cd04, /ptr/colr/hue, print).$

%the second path combination,policies P1, P4
$\langle(n, 5, 3, p)\rangle\ auth(/doc/dse/stud/cd04, /ptr/huxbldg/lv5/hue, print).$
$\langle(f, 3, 2, p)\rangle\ auth(/doc/dse/stud/cd04, /ptr/huxbldg/lv5/hue, print).$

%the third path combination,policies P1, P2, P3
$\langle(n, 5, 3, p)\rangle\ auth(/doc/stud/phd/cd04, /ptr/colr/hue, print).$
$\langle(n, 3, 2, p)\rangle\ \neg auth(/doc/stud/phd/cd04, /ptr/colr/hue, print).$
$\langle(n, 2, 1, p)\rangle\ auth(/doc/stud/phd/cd04, /ptr/colr/hue, print).$

%the fourth path combination,policy P1
$\langle(n, 6, 3, p)\rangle\ auth(/doc/stud/phd/cd04, /ptr/huxbldg/lv5/hue, print).$

Fig. 3. Labelled Rules for the Example

domain */ptr* is 2, therefore *tdis* is the sum, 5, and *sdis* is 3. $P5$ is a negative authorisation policy, so the labelled rule is with a label $\langle(n, 2, 1, n)\rangle$ and head $\neg auth(/doc/dse/stud/cd04, /ptr/colr/hue, print)$. In the same way, $P6$ is translated into a labelled rule with a label $\langle(n, 1, 1, p)\rangle$.

4.3 Conflict Resolution Strategy

Although policies defined over hierarchies simplify configuration and management they also give rise to conflicts. In Ponder2, when multiple policies along the path from the subject to the target have different signs, conflict occurs. Ponder2 resolves these types of conflict using the following rules: (1) the most specific policy takes precedence; (2) if two policies are equally specific, the negative one takes precedence. Informally, a policy that applies to a subdomain is more specific than a policy that applies to any ancestor domains. The specificity is determined by the distance from the subject to the target using the policy arc. In case 4-(a), the path from the subject to the target through $p1$ is "s, c, a, d, t" and the path from the subject to the target through $p2$ is "s, c, d, t". The second path is shorter, so $p2$ is more specific than policy $p1$. Case 4-(b) shows a not so intuitive example where both paths have the same length. In such cases, Ponder2 gives higher importance to the subject side path. Therefore, $p2$ is more specific than $p1$ because domain c is closer to s than domain a. Case 4-(c) shows an example where two paths have the same length and are defined between the same levels, i.e. they are equally specific. According to rule (2), $p1$ "wins" because it is negative.

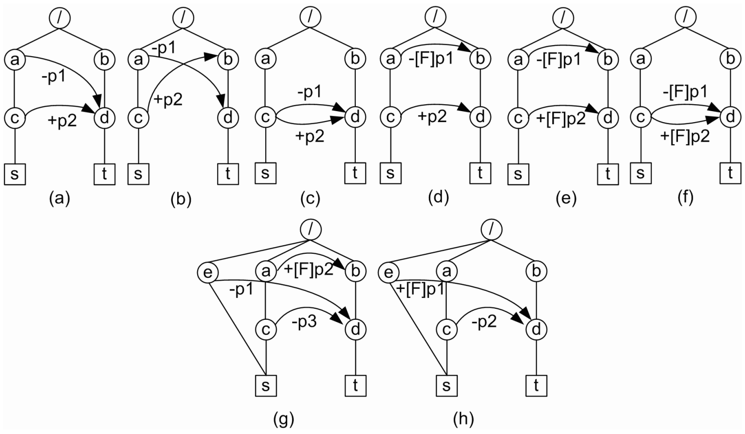


Fig. 4. Authorisation conflicts examples

Sometimes it is desirable that a general policy overrides more specific ones. When a policy is defined as a **final** policy, it has a higher priority than the normal ones. If more than one final policy exists, the conflict resolution rules for final policies are: (1) the most general final policy takes precedence; (2) if two policies are equally specific, the negative one takes precedence. In figure 4, final policies are shown with a prefix **[F]**. In case 4-(d), a final policy "wins" even though a normal policy $p2$ is more specific. In case 4-(e), the more general final policy $p1$ overrides a more specific final policy $p2$. In case 4-(f), a final policy $p1$ overrides another final policy $p2$ because they are equally general and $p1$ is negative.

When there are multiple domain paths from a MO s to MO t, the domain nesting rules cannot be applied because the policies are defined over different paths. In such situations, the path authorisation for each path combination is derived using the above rules. The overall authorisation is then determined by aggregating the path authorisations. If one of the path authorisations is negative, then the result is negative; if all the path authorisations are positive, then the result is positive. In the examples shown in Figure 4-(g), (h), there are two path combinations: $(/e/s, /b/d/t)$ and $(/a/c/s, /b/d/t)$. In case 4-(g), the path authorisation for the first combination is negative and for the second combination is positive, therefore the overall authorisation is negative. In case 4-(h), the overall authorisation is also negative because there is a negative path authorisation. Note that the final keyword only affects conflict resolution in domain nesting cases. A final policy cannot override a normal policy when the normal policy is defined over other path combinations.

Administrators are required to define a default authorisation, either positive or negative that is applied when an authorisation request has no applicable policy.

4.4 CLP Conflict Resolution Strategy for Example

The conflict resolution rules for Ponder2 can be captured as a small overrides sub-program in CLP that is used to prioritise the labels and resolve any conflicts. Recall that we generated two kinds of labels when we translated Ponder2 policies into CLP.

For the first set of labels (p),(n),(d), we define the following overrides rules:

$$overrides((n), (p)). \tag{1}$$
$$overrides((p), (d)). \tag{2}$$
$$overrides((n), (d)). \tag{3}$$

The meaning of the above rules is that a negative path authorisation has the highest priority, then the positive ones, and finally the default authorisation which has the lowest priority.

For the set of labels of the form $(type, dis1, dis2, mod)$, we have the following overrides rules. First,

$$overrides((f, \_, \_, \_), (n, \_, \_, \_)) \tag{4}$$

states that the priority of a final policy is always higher than a normal one. Then among the final policies, the overrides rules are:

$$overrides((f, X1, \_, \_), (f, X2, \_, \_)) \leftarrow X1 > X2. \tag{5}$$
$$overrides((f, X, Y1, \_), (f, X, Y2, \_)) \leftarrow Y1 > Y2. \tag{6}$$
$$overrides((f, X, Y, n), (f, X, Y, p)). \tag{7}$$

which state that a more general final policy always has more priority, and that we always give higher priority to a negative policy if there is a "tie".

For normal policies, the rules are reversed and the more specific policy wins:

$$overrides((n, X1, \_, \_), (n, X2, \_, \_)) \leftarrow X1 < X2. \tag{8}$$
$$overrides((n, X, Y1, \_), (n, X, Y2, \_)) \leftarrow Y1 < Y2. \tag{9}$$
$$overrides((n, X, Y, n), (n, X, Y, p)). \tag{10}$$

It is easy to see that the *overrides* relations defined in the sub-program are transitive, anti-symmetric and irreflexive, and therefore meet the requirement.

4.5 Resolving Conflicts in CLP

We now explain how CLP resolves conflicts for our example. For path combination (p_{s1}, p_{t1}), i.e. $(/doc/dse/stud/cd04, /ptr/colr/hue, print)$, there are two labelled rules which permit the action. The labels of these two rules are $\{(n, 5, 3, p), (n, 1, 1, p)\}$. There is also one rule which denies the action, whose label is $\{(n, 2, 1, n)\}$. It's clear that given the conflict resolution strategy in Section 4.4, the following are true:

$$overrides((n, 2, 1, n), (n, 5, 3, p)).$$
$$overrides((n, 1, 1, p), (n, 2, 1, n)).$$

Although the negative rule overrides one of the positive rules, it itself is also overridden by another positive rule. Therefore $auth(/doc/dse/stud/cd04, /ptr/colr/hue, print)$ is true because a positive rule has the highest priority. In the same way, we get the following for the other path combinations:

$$auth(/doc/dse/stud/cd04, /ptr/huxBldg/lv5/hue).$$
$$auth(/doc/stud/phd/cd04, /ptr/colr/hue, print).$$
$$auth(/doc/stud/phd/cd04, /ptr/huxBldg/lv5/hue).$$

Once the authorisation status of each path combination has been decided, the overall authorisation result can be decided. Given the above results, we can derive $auth(cd04, hue, print)$ from the CLP program and this is derived from rules labelled $\langle(p)\rangle$. Although a conflicting authorisation, $\neg auth(cd04, hue, print)$, can also be derived from the default rule, the label of the default rule, $\langle(d)\rangle$, has a lower priority than $\langle(p)\rangle$. Finally the request is authorised and cd04 can print on hue.

5 Alternative Conflict Resolution Strategies

To support different resolution strategies, we simply change the prioritisation rules (the overrides sub-program). In the example negative authorisations takes precedence. If we want to change this to positive takes precedence, we only need to modify rules (1), (7), (10) in Section 4.4 into:

$$overrides((p), (n)).$$
$$overrides((f, X, Y, p), (f, X, Y, n)).$$
$$overrides((n, X, Y, p), (n, X, Y, n)).$$

The most general final policy takes precedence can be changed to the most specific final policy takes precedence by modifying rules (5),(6) into:

$$overrides((f, X1, \_, \_), (f, X2, \_, \_)) \leftarrow X1 < X2.$$
$$overrides((f, X, Y1, \_), (f, X, Y2, \_)) \leftarrow Y1 < Y2.$$

In this case, we can even combine rules (5), (6), (8), (9) into the following two rules:

$$overrides((W, X1, \_, \_), (W, X2, \_, \_)) \leftarrow X1 < X2.$$
$$overrides((W, X, Y1, \_), (W, X, Y2, \_)) \leftarrow Y1 < Y2.$$

Ponder2 defines the most specific policy as the one with the shortest path from the subject to the target and in the case that there are several ones with the same path length, the one which is closest to the subject. We can change this to allow the one closest to the target to win. In the context of the most general final/most specific normal policy takes precedence, this can be done by modifying rules (6), (9) into:

$$overrides((f, X, Y1, \_), (f, X, Y2, \_)) \leftarrow Y1 < Y2.$$
$$overrides((n, X, Y1, \_), (n, X, Y2, \_)) \leftarrow Y1 > Y2.$$

After modification, if two policies have the same distance, the one closer to the target takes precedence. Many other variations are possible as long as the override rules define a strict partial order.

Besides assigning priorities according to the domain hierarchy and inheritance, our approach can easily support conflict resolution strategies based on other factors. For example, which administrator defined this policy, who the owner of a policy is, the date and time that a policy was enabled, etc. The translation module needs to obtain all such information when an access request is made and include the information in the labels of CLP authorisation rules. Strategies would then assign priorities according to the new labels.

6 Related Work

Early authorisation models such as SeaView [13] and the Andrew File System [14] employ a simple negative takes precedence rule to resolve conflicts. However, as the authorisation models become more complex, such rules are not enough for handling all the conflicts.

Woo and Lam [15,7] propose an access control model for distributed systems where the management of authorisation is decentralised. The authorisation requirements are specified as policy rules using a language similar to default logic. Conflicts can be resolved by either positive takes precedence or negative takes precedence. The problem of this model is that it has no hierarchical structure for organising subjects or objects. Therefore, all the policies must be defined instance by instance, which will be burdensome in large systems.

Bertino et.al. [3] propose an authorisation model for relational data management systems. In this model, subjects are grouped in a group hierarchy and authorisation policies are classified as *strong* and *weak*. Strong policies always override weak policies and conflicts among strong policies are not allowed. The administrators must be very careful to avoid the conflicts among strong policies. The conflict resolution strategy is fixed in this model, while in our framework, the resolution strategy can be tailored to meet different applications' requirements.

Jajodia et.al. [6] proposed a flexible authorisation framework and a formal language called Authorisation Specification Language (ASL). The subjects are organised into hierarchies and the authorisation is defined as rows in an authorisation table. To define a full-fledged conflict resolution strategy, an administrator needs to define the following: a propagation policy that specifies how to derive authorisations according to the hierarchies and the authorisation table; a conflict resolution policy that specifies how to eliminate conflicts; a decision policy that decides the default authorisation in the absence of explicit specifications for the access; a set of integrity constraints that impose restrictions on the content and the output. The conflict resolution in this framework is quite flexible and powerful, but complex to understand and support. Our approach is much simpler. All the administrator needs to do is to decide an ordering over policies based on the properties captured by the labels (modality, position in the hierarchy and so on), and supply a small override program.

Benferhat et.al. [16] proposed a stratification-based approach for handling conflicts in access control. They classify the information used in access control as facts, rules without exceptions, and rules with exceptions. The information is prioritised as follows: facts and rules without exceptions are always preferred to rules with exceptions; for rules with exceptions, more specific ones are preferred to the more general ones. After information has been stratified according to the priority, conflicts can be solved by probabilistic logic inference or lexicographical inference. However, this approach requires that the facts and the rules without exceptions to be consistent and can only implement one conflict resolution strategy for the rules with exceptions.

Chadha [17] argued that many application-specific runtime policy conflicts can be addressed by re-writing policies. Rather than writing policies and defining complex resolution rules, it may be simpler to just re-write the policies. The conclusions were based on the analysis of obligation policies and conflicts such as redundant conflicts, mutually exclusive configurations or inconsistent configuration. Authorisation policies and modality conflicts, which is the focus of our work, were not discussed in detail. However, the author also concluded that "conflicts such as modality conflicts that are application-independent should be resolved automatically using conflict resolution tools", which is exactly what our approach trying to do.

7 Conclusions and Future Work

In this paper we express conflict resolution strategies in terms of Courteous Logic Programs. Authorisations are dynamically translated into labelled logic rules. The label for each rule is determined by a translation module specific to each authorisation model and can use features such as the policy type, the domain hierarchy and the policy modality. The priority of the label is defined through a special *overrides* predicate and conflicts are resolved by choosing the rule with the highest priority. The conflict resolution rules are not static and can be dynamically changed according to domain-specific requirements.

The Ponder2 policy model was used as a case study and to demonstrate a variety of conflict resolution strategies and how easy it is for administrators to define their own. An implementation of our CLP translation module and PDP is available as an option for Ponder2 and can be used instead of its Java-coded conflict resolution strategy. The implementation is based on a Prolog version of CLP engine. We hope to develop an implementation and a library of conflict resolution strategies for XACML (Extensible Access Control Markup Language) [18] in the near future, and to investigate how to combine multiple strategies.

References

1. Feridun, M., Leib, M., Nodine, M.H., Ong, J.: Anm: Automated network management system. IEEE Network 2(2), 13–19 (1988)
2. Strassner, J.: Policy-based network management, solution for the next generation. Morgan and Kaufmann, San Francisco (2004)
3. Bertino, E., Jajodia, S., Samarati, P.: A flexible authorization mechanism for relational data management systems. ACM Trans. Inf. Syst. 17(2), 101–140 (1999)
4. Bertino, E., Samarati, P., Jajodia, S.: Authorizations in relational database management systems. In: ACM Conference on Computer and Communications Security, pp. 130–139 (1993)
5. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: POLICY, pp. 18–38 (2001)
6. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. ACM Trans. Database Syst. 26(2), 214–260 (2001)
7. Woo, T.Y.C., Lam, S.S.: Authorizations in distributed systems: A new approach. Journal of Computer Security 2(2-3), 107–136 (1993)
8. Lupu, E., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Trans. Software Eng. 25(6), 852–869 (1999)
9. Grosof, B.N.: Courteous logic programs: Prioritized conflict handling for rules. Research Report RC 20836(92273), IBM (1997)
10. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
11. Ponder2: The Ponder2 project. `www.ponder2.net`
12. Russello, G., Dong, C., Dulay, N.: Authorisation and conflict resolution for hierarchical domains. In: POLICY, pp. 201–210 (2007)
13. Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M., Shockley, W.R.: The seaview security model. IEEE Trans. Software Eng. 16(6), 593–607 (1990)
14. Satyanarayanan, M.: Integrating security in a large distributed system. ACM Trans. Comput. Syst. 7(3), 247–280 (1989)
15. Woo, T.Y.C., Lam, S.S.: Authorization in distributed systems: A formal approach. In: SP 1992: Proceedings of the 1992 IEEE Symposium on Security and Privacy, pp. 33–50. IEEE Computer Society, Washington (1992)
16. Benferhat, S., Baida, R.E., Cuppens, F.: A stratification-based approach for handling conflicts in access control. In: SACMAT, pp. 189–195 (2003)
17. Chadha, R.: A cautionary note about policy conflict resolution. In: Military Communications Conference, 2006. MILCOM 2006, pp. 1–8 (October 2006)
18. XACML: Extensible access control markup language. `http://xml.coverpages.org/xacml.html`

Trust Management for Host-Based Collaborative Intrusion Detection

Carol J. Fung, Olga Baysal, Jie Zhang, Issam Aib, and Raouf Boutaba

David R. Cheriton School of Computer Science
University of Waterloo, Canada
{j22fung,obaysal,j44zhang,iaib,rboutaba}@uwaterloo.ca

Abstract. The accuracy of detecting an intrusion within a network of intrusion detection systems (IDSes) depends on the efficiency of collaboration between member IDSes. The security itself within this network is an additional concern that needs to be addressed. In this paper, we present a trust-based framework for secure and effective collaboration within an intrusion detection network (IDN). In particular, we define a trust model that allows each IDS to evaluate the trustworthiness of others based on personal experience. We prove the correctness of our approach in protecting the IDN. Additionally, experimental results demonstrate that our system yields a significant improvement in detecting intrusions. The trust model further improves the robustness of the collaborative system against malicious attacks.

Keywords: Intrusion detection Network, Trust Management, Collaboration, Peer-to-Peer, Security.

1 Introduction

Intrusions over the Internet are becoming more dynamic and sophisticated. Intrusion Detection Systems (IDSes) identify intrusions by comparing observable behavior against suspicious patterns. They can be network-based (NIDS) or host-based (HIDS). Traditional IDSes work in isolation and may be easily compromised by unknown or new threats. An Intrusion Detection Network (IDN) is a collaborative IDS network intended to overcome this weakness by having each members IDS benefit from the collective knowledge and experience shared by other member IDSes. This enhances their overall accuracy of intrusion assessment as well as the ability of detecting new intrusion types.

Intrusion types include worms, spamware, viruses, denial-of-service(DoS), malicious logins, etc. The potential damage of these intrusions can be significant if they are not detected promptly. An example is the Code Red worm that infected more than 350,000 systems in less than 14 hours in 2001 with a damage cost of more than two billion dollars [7]. IDS collaboration can also be an effective way to throttle or stop the spread of such contagious attacks.

Centralized collaboration of IDSes relies on a central server to gather alerts and analyze them. This technique suffers from the performance bottleneck problem. In addition the central server is a single point of failure and may become the target of denial-of-service attacks. The distributed collaboration of IDSes can avoid these problems. However, in such a collaborative environment, a malicious IDS can degrade the performance of others by sending out false evaluations about intrusions. To protect an IDS collaborative network from malicious attacks, it is important to evaluate the trustworthiness of participating IDSes, especially when they are Host-based IDSes (HIDSes). Duma et al. [3] propose a simple trust management model to identify dishonest insiders. However, their model is vulnerable to some attacks which aim at compromising the trust model itself (see Sections 4 and 5 for more details).

In this work, we develop a robust trust management model that is suitable for distributed HIDS collaboration. Our model allows each HIDS to evaluate the trustworthiness of others based on its own experience with them. We also propose a framework for efficient HIDS collaboration using a peer-to-peer network. Our framework provides identity verification for participating HIDSes and creates incentives for collaboration amongst them.

We evaluate our system based on a simulated collaborative HIDS network. The HIDSes are distributed and may have different expertise levels in detecting intrusions. A HIDS may also become malicious in case it has been compromised (or the HIDS owner deliberately makes it malicious). We also simulate several potential threats. Our experimental results demonstrate that our system yields a significant improvement in detecting intrusions and is robust to various attacks as compared to that of [3].

The rest of the paper is organized as follows. Section 2 presents the collaborative IDS framework and collaboration management mechanisms. Section 3 formalizes our trust management model. Section 4 addresses common attacks on the collaboration and proves how our trust model cancels them. Section 5 presents the different simulation settings used and discusses the obtained results. Section 6 discusses related work. Finally, Section 7 summarizes our contributions and addresses future work directions.

2 HIDS Collaboration Framework

The purpose of this framework is to connect individual HIDSes so that they can securely communicate and cooperate with each other to achieve better intrusion detectability. Figure 1 illustrates the key components of our framework. Collaboration is ensured by trust-based cooperation and peer-to-peer communication. The trust management model allows a HIDS to evaluate the trustworthiness of its neighbors based on its own experience with them. The P2P component provides network organization, management and communication between the HIDSes. The collaboration is ensured by three processes, which are explained in the following.

2.1 Network Join Process

In our framework, each HIDS connects to other HIDSes over a peer-to-peer network. Before joining the network, a HIDS node needs to register to a trusted

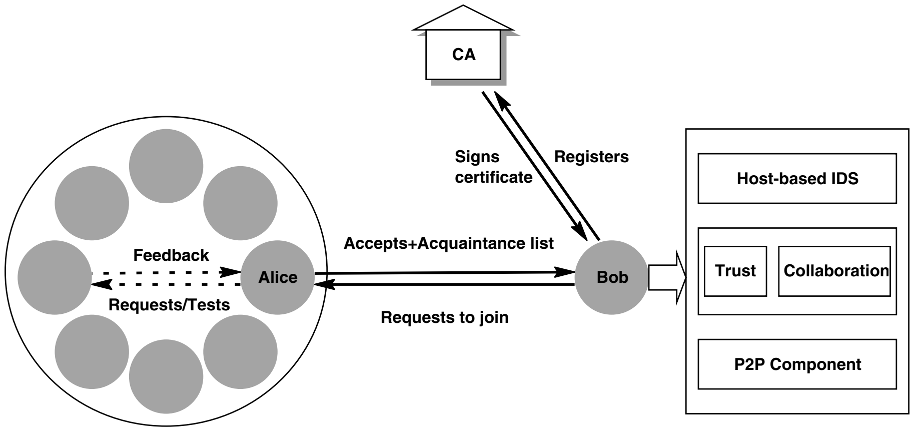


Fig. 1. IDS Collaboration Framework

digital certificate authority (Figure 1) and get a public and private key pair which uniquely identifies it. Note that we identify the (machine, user) tuple. This is because a different machine means a different HIDS instance. In addition, a different user of the same machine may have a different configuration of its HIDS. After a node joins the HIDS network, it is provided with a preliminary *acquaintance list*. This list is customizable and contains IDs (or public keys) of other nodes within the network along with their *trust* values and serves as the contact list for collaboration.

2.2 Test Messages

Each node sends out either *requests* for alert ranking (consultation), or *test messages*. A test message is a consultation request sent with the intention to evaluate the trustworthiness of another node in the acquaintance list. It is sent out in a way that makes it difficult to be distinguished from a real alert ranking request. The testing node knows the *severity* of the alert described in the test message and uses the received feedback to derive a trust value for the tested node. This technique helps in uncovering inexperienced and/or malicious nodes within the collaborative network.

2.3 Incentive Design

Our framework also provides *incentives* to motivate collaboration. Nodes that are asked for consultation will reply to only a number of requests in a certain period of time because of their limited bandwidth and computational resources. Thus, only highly trusted nodes will have higher priority of receiving help whenever needed. In this way, nodes are encouraged to build up their trust. In addition, our system accepts the reply of "*don't know*" to requests in order to encourage active collaboration in the network. This is explained in section 3.1.

3 Trust Management Model

This section describes the model we developed to establish trust relationships between the HIDSes in the collaborative environment. We first describe how we evaluate the trustworthiness of a HIDS and then present a method to aggregate feedback responses from trusted neighbours.

3.1 Evaluating the Trustworthiness of a Node

The evaluation of the trustworthiness of a node is carried out using test messages sent out periodically using a random poisson process. After a node receives the feedback for an alert evaluation it assigns a satisfaction value to it, which can be "very satisfied" (1.0), "satisfied" (0.5), "neutral" (0.3), "unsatisfied" (0.1), or "very unsatisfied" (0).

The trust value of each node will be updated based on the satisfactory levels of its feedback. More specifically, the replies from a node i are ordered from the most recent to the oldest according to the time t_k at which they have been received by node j. The *trustworthiness* of node i according to node j can then be estimated as follows:

$$tw_i^j(n) = \frac{\sum_{k=0}^{n} S_k^{j,i} F^{t_k}}{\sum_{k=0}^{n} F^{t_k}} \tag{1}$$

where $S_k^{j,i} \in [0,1]$ is the satisfaction of the reply k and n is the total number of feedback. To deal with possible changes of the node behavior over time, we use a *forgetting factor* F $(0 \leq F \leq 1)$ which helps in assigning less weight to older feedback responses [10]. Compared to Duma et al. [3], our model uses multiple satisfaction levels and forget old experiences exponentially, while [3] only uses two satisfaction levels (satisfied and unsatisfied) and all experiences have the same impact.

We also allow a node to send a "don't know" answer to a request if it has no experience with the alert or is not confident with its ranking decision. However, some nodes may take advantage of this option by always providing "don't know" feedback responses so as to maintain their trust values. In order to encourage nodes to provide satisfactory feedback responses whenever possible, the trust value will be slowly updated every time the node provides a "don't know" answer. The trustworthiness of a node i according to node j is then formulated as follows:

$$T_i^j = (tw_i^j - T_{stranger})(1-x)^m + T_{stranger}, \tag{2}$$

where x is the percentage of "don't know" answers from time t_0 to t_n. m is a positive incentive parameter (*forgetting sensitivity*) to control the severity of punishment to "don't know" replies, tw_i^j is the trust value without the integration of "don't know" answers (Equation 1), and $T_{stranger}$ is the default trust

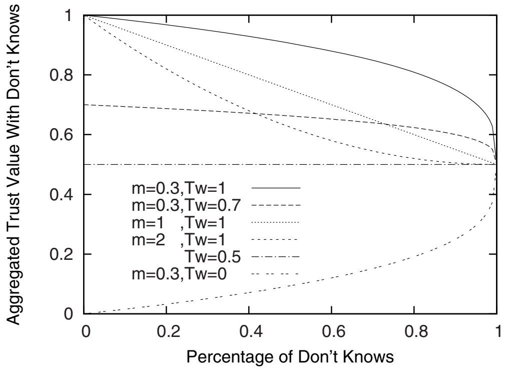


Fig. 2. Trust Convergence Curves with "don't know" answers

value of a stranger. As illustrated in Figure 2, Equation 2 causes trust values to converge to $T_{stranger}$ with the increase in the percentage of don't "know answers". Eventually, the trust value will become that of a stranger. This allows the trust value of an untrusted node to slowly increase up to the level of a stranger by providing "don't know" answers. In addition, nodes with little experience are motivated to provide "don't know" answers rather than incorrect alert rankings.

3.2 Feedback Aggregation

Based on the history of trustworthiness, each node i requests alert consulting only from those nodes in its acquaintance list whose trust values are greater than a threshold th_t^i. We also consider *proximity*, which is a measure of the physical distance between the node that provides feedback and the node that sends the request. Physical location can be an important parameter in intrusion detection. HIDSes that are located in the same or close by geographical region are likely to experience similar intrusions [1] and thus can help each other by broadcasting warnings of active threats. Feedback from nearby acquaintances is therefore more relevant than that from distant ones. We scale the proximity based on the region the node belongs to.

After receiving feedback from its acquaintances, node j aggregates the feedback using a *weighted majority* method as follows:

$$R_j(a) = \frac{\sum\limits_{T_i^j \geq th_t^j} T_i^j D_i^j R_i(a)}{\sum\limits_{T_i^j \geq th_t^j} T_i^j D_i^j}, \tag{3}$$

where $R_j(a)$ is the aggregated ranking of alert a from the feedback provided by each node belonging to the acquaintance list A_j of node j. T_i^j ($\in [0,1]$) is the trust value of node i according to node j. D_i^j ($\in [0,1]$) is the proximity weight of node i. th_t^j is the trust threshold set by node j. $R_i(a)$ ($\in [0,1]$) is the feedback ranking of alert a by node i.

Compared to [3], our model only integrate feedback from trusted nodes while [3] integrates feedback from all neighbors.

4 Robustness Against Common Threats

Trust management can effectively improve network collaboration and detect malicious HIDSes. However, the trust management itself may become the target of attacks and be compromised. In this section, we describe possible attacks and provide defense mechanisms against them.

Sybil attacks occur when a malicious node in the system creates a large amount of pseudonyms (fake identities) [2]. This malicious node uses fake identities to gain larger influence of the false alert ranking on others in the network. Our defense against sybil attacks relies on the design of the authentication mechanism. Authentication makes registering fake IDes difficult. In our model, the certificate issuing authority only allows one ID per IP address. In addition, our trust management model requires IDSes to first build up their trust before they can affect the decision of others, which is costly to do with many fake IDes. Thus, our security and trust mechanisms protect our collaborative network from sybil attacks.

Identity cloning attacks occur when a malicious node steals some node' identity and tries to communicate with others on its behalf. Our communication model is based on asymmetric cryptography, where each node has a pair of public and private keys. The certificate authority certifies the ownership of key pairs and in this way protects the authenticity of node identities.

Newcomer attacks occur when a malicious node can easily register as a new user [8]. Such a malicious node creates a new ID for the purpose of erasing its bad history with other nodes in the network. Our model handles this type of attack by assigning low trust values to all newcomers, so their feedback on the alerts is simply not considered by other nodes during the aggregation process.

Betrayal attacks occur when a trusted node suddenly turns into a malicious one and starts sending false alerts or even malware. A trust management system can be degraded dramatically because of this type of attacks. We employ a mechanism which is inspired by the social norm:

> *It takes a long-time interaction and consistent good behavior to build up a high trust, while only a few bad actions to ruin it.*

When a trustworthy node acts dishonestly, the forgetting factor (Eqn.1) causes its trust value to drop down quickly, hence making it difficult for this node to deceive others or gain back its previous trust within a short time.

Collusion attacks happen when a group of malicious nodes cooperate together by providing false alert rankings in order to compromise the network. In our system, nodes will not be adversely affected by collusion attacks. In our trust model each node relies on its own knowledge to detect dishonest nodes.

In addition, we use test messages to uncover malicious nodes. Since the test messages are sent in a random manner, it will be difficult for malicious nodes to distinguish them from actual requests.

5 Simulations and Experimental Results

In this section, we present the experiments used to evaluate the effectiveness and robustness of our trust-based IDS collaboration framework.

5.1 Simulation Setting

In our simulation model, we have n nodes in the collaboration network randomly distributed in a $s \times s$ grid region. The proximity weight of nodes is anti-proportional to the distance between the nodes in the number of grid steps. The expertise levels of the nodes can be low(0.1), medium(0.5) or high(0.95). In the beginning, each node builds an initial acquaintance list based on the communication cost (proximity). The initial trust values of all nodes in the acquaintance list are set to the stranger trust value ($T_{stranger}$). To test the trustworthiness of all the acquaintances in the list, each node sends out test messages following a Poisson process with average arrival rate λ_t. The intrusion detection expertise of a HIDS is modeled using a beta function. A honest HIDS always generates feedback based on its truthful judgment, while a dishonest HIDS always sends feedback opposite to its truthful judgment. The parameters used in the simulation are shown in Table 1.

To reflect a HIDS expertise level, we use a beta distribution for the decision model of HIDSes. The beta density function is expressed in Eqn.5:

$$f(p|\alpha, \beta) = \frac{1}{B(\alpha, \beta)} p^{\alpha-1}(1-p)^{\beta-1}, \tag{4}$$

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt, \tag{5}$$

Table 1. Simulation Parameters for Experiments

Parameter	Value	Description
λ_t	5/day	Test messages frequency
F	0.9	Forgetting factor
$T_{stranger}$	0.5	Stranger trust value
th_t	0.8	Trust threshold
m	0.3	Forgetting sensitivity
Th_{DK}	1	Threshold of “don’t know” replies
s	4	Size of grid region
n	30 (+10)	Number of HIDS

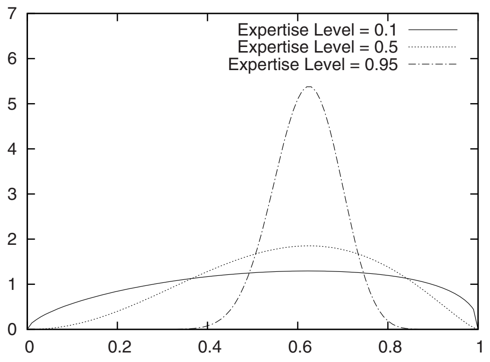


Fig. 3. Decision DF for Expertise Levels

We define α and β as:

$$\alpha = 1 + \frac{(1-D)}{D}\sqrt{\frac{E(b-1)}{1-E}}, \qquad \beta = 1 + \frac{(1-D)(1-E)}{DE}\sqrt{\frac{E(b-1)}{1-E}} \tag{6}$$

where E is the expected ranking of the alert. D ($\in [0,1]$) denotes the difficulty level of a alert. Higher values for D are associated to attacks that are difficult to detect, i.e. many HIDSes fail to identify them. $L(\in [0,1])$ denotes the expertise level of an IDS. A higher value for L reflects a higher probability of producing correct rankings for alerts. $f(p;\alpha,\beta)$ is the probability that the node with expertise level L answers with a value of p $(1 \geq p \geq 0)$ to an alert of difficulty level D, $\alpha \geq 1$, $\beta \geq 1$, and $b = 1/(1-L)^2$. For a fixed difficulty level, this model assigns higher probabilities of producing correct ranking to nodes with higher levels of expertise. For a node with fixed expertise level, it has a lower probability of producing correct rankings for alerts with higher D values. A node with expertise level 1 or an alert with difficulty level 0 represents the extreme case that the node can rank the alert accurately with guarantee. This is reflected in the Beta distribution by parameters $\alpha = \infty$ and $\beta = \infty$. A node with expertise level 0 or an alert with difficulty level 1 represents the extreme case that the node ranks the alert by picking up answer randomly. This is reflected in the Beta distribution by parameters $\alpha = 1$ and $\beta = 1$ (Uniform distribution). Figure 3 shows the feedback probability distribution for IDSes with different expertise levels, where the expected risk level is fixed to 0.7 and the difficulty level of test messages is 0.5.

5.2 Results for a Honest Environment

The first experiment studies the effectiveness of IDS collaboration and the importance of trust management. In this experiment, all IDSes are honest. 30 IDSes are divided into three equally-sized groups, with expertise levels of 0.1, 0.5 and 0.95 respectively. We simulate the first 100 days to observe the trust values

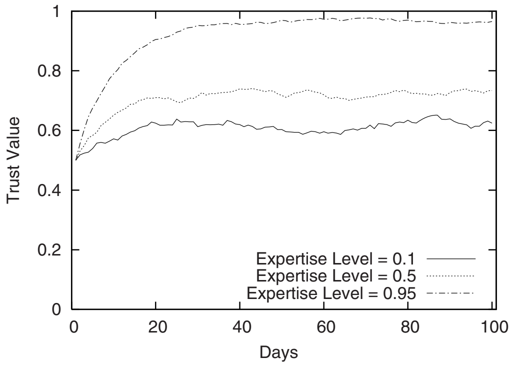


Fig. 4. Convergence of Trust Values for Different Expertise Levels during the learning period

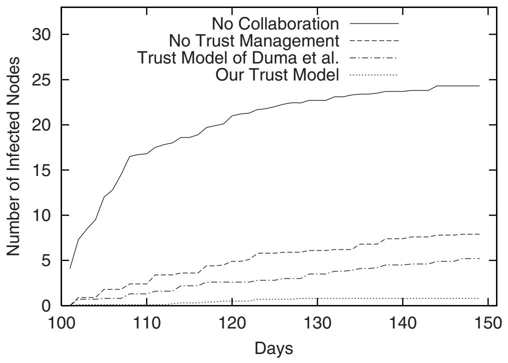


Fig. 5. Number of Infected Nodes for a Honest Environment during the attack period

of nodes from each group, where nodes send only test messages to the others. Figure 4 shows the average trust values of nodes with different expertise levels. We can see that after 40-50 days, the trust values of all nodes converges to stable values.

Starting from day 101, we inject one random attack to all the nodes in the network in each day. The risk values of the attacks are uniformly generated from [low, medium, high]. The difficulty levels of attacks are fixed to 0.5. Each IDS in the network ranks the alert generated by the attack. If a node ranks "no risk" or "low risk" for a high-risk attack, then it is assumed to have been infected. We observe the total number of infected nodes in the network from day 101 to day 150 under different collaboration modes: no collaboration, collaboration without trust management, the trust management adapted from the model of Duma et al. [3] and our trust management method. The results of the total number of infected nodes are shown in Figure 5. In this figure, we can see that the network in collaboration mode is more resistant to attacks than the network in

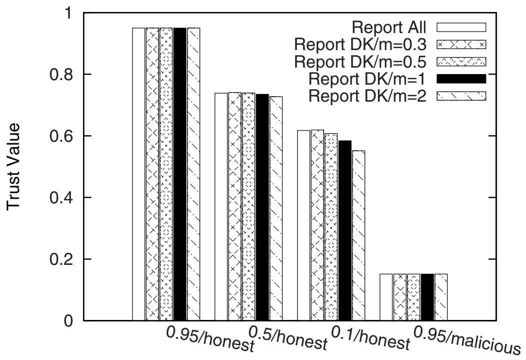


Fig. 6. Converged Trust Values for Different Expertise/Honesty Levels

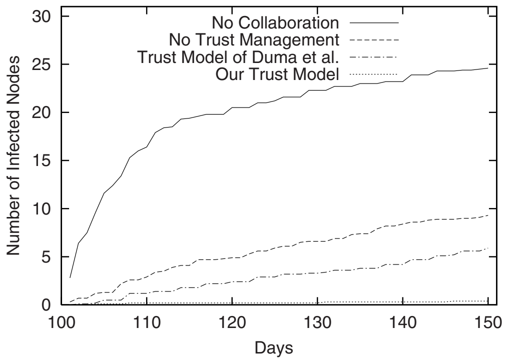


Fig. 7. Number of Infected Nodes for Dishonest Environment

non-collaborative mode. Trust management models further improve the effectiveness of the collaboration. Our trust model performs better than that of Duma et al. [3]. Almost all attacks are detected and almost no node is infected after 50 days.

5.3 Results for an Environment with Some Dishonest Nodes

The purpose of the second experiment is to study the effectiveness of the collaboration model in a hazard situation where some nodes in the network are dishonest. We look at a special case where only some expert nodes are dishonest because malicious expert nodes have the largest impact on the system.

In this experiment, we have 10 expert nodes that are dishonest. There are two cases, without and with "don't know" replies. In the latter case, the percentages of "don't know" answers from nodes with expertise levels of 0.95, 0.5 and 0.1 are 0%, 4% and 45% respectively. The forgetting sensitiveness parameter (m) varies from 0.3 to 2. Figure 6 shows the converged trust values of nodes with

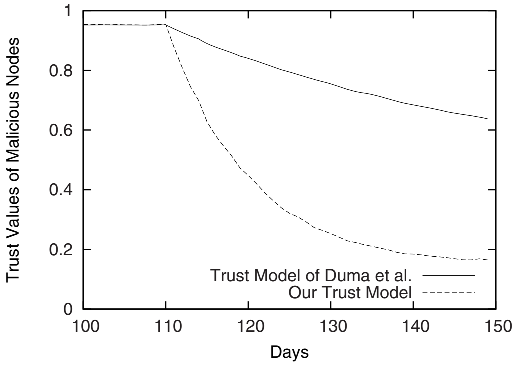


Fig. 8. Trust of Malicious Nodes

different expertise/honesty levels in "report all" case and "report don't know" case after 100 simulation days. When m is large, the punishment of reporting "don't know" is heavier. For $m = 0.3$, the nodes with medium (0.5) and low (0.1) expertise levels will be slightly rewarded. Therefore, we suggest using $m = 0.3$ to encourage non-expert nodes to report "don't know" when they are not confident about their replies.

After 100 days, we start injecting randomly generated attacks to all the nodes in the network at a rate of one attack per day. Figure 7 shows the total number of infected nodes under different collaboration modes. The number of infected nodes in the no-collaboration case is about the same as that in Figure 5 because the HIDSes make decisions independently. When there is no trust model or using the model of Duma et al. [3] to detect malicious nodes, the total number of infected nodes is larger than the corresponding case in Figure 5. The network hence suffers from malicious nodes. The number of infected nodes remains very small when using our trust model. This shows how important effective trust management is for a HIDS collaboration system.

5.4 Robustness of the Trust Model

The goal of this experiment is to study the robustness of our trust model against attacks. For the newcomer attack, malicious nodes white-wash their bad history and re-register as new users to the system. However, a newcomer attack is difficult to succeed in our system. This is because it takes a long time for a newcomer to gain trust over the trust threshold. In our experiment, it takes about 15 days for an expert node to gain trust of 0.8 to pass the threshold (as shown in Figure 4).

The second possible threat is the betrayal attack, where a malicious node gains a high trust value and then suddenly starts to act dishonestly. This scenario happens, for example, when a node is compromised. To demonstrate the robustness of our model against this attack type, we add 10 expert nodes which spread opposite alert rankings on day 110. Figures 8 and 9 show the trust

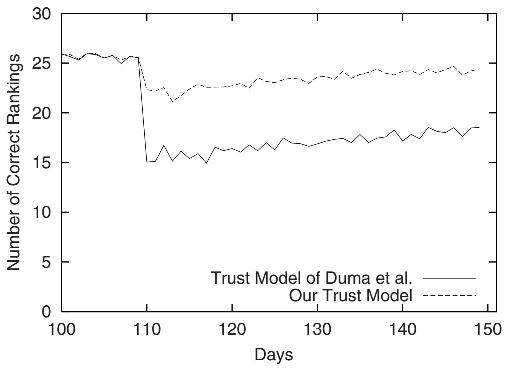


Fig. 9. The Impact of Betrayal Attack

values of the betraying nodes and the number of correct attack rankings before and after the betrayal attack when using the trust model of Duma et al. and our trust model respectively. When using our trust model, we notice that the impact of the betrayal attack is smaller and the trust of malicious nodes drops down faster. This is because our trust model uses a forgetting factor. The trust values of dishonest nodes rely more on recent experience and therefore decrease more quickly. Our recovery phase is also shorter because we use a threshold to eliminate the impact of dishonest nodes. Once the trust values of malicious nodes drops under the trust threshold of 0.8, they are ignored in the alert consultation process and their impact is completely eliminated.

6 Related Work

Most of the existing work on distributed collaborative intrusion detection relies on the assumption that all IDSes are trusted and faithfully report intrusion events. For instance, [4] proposes a distributed information sharing scheme among trusted peers to guard against intrusions. and [6] proposes a distributed intrusion detection system based on the assumption that all peers are trusted. However, both systems can be easily compromised when some of the peers are (or become) dishonest. Duma et al. [3] address possibly malicious peer IDSes by introducing a trust-aware collaboration engine for correlating intrusion alerts. Their trust management scheme uses each past experience of a peer to predict the trustworthiness of other peers. However, their trust model is naive and does not address security issues within the collaborative network. For instance, in their system, the past experience of a peer has the same impact on its final trust values regardless of the age of its experience, therefore making it vulnerable to betrayal attacks. In our model, we use a forgetting factor when computing the trust to put more emphasis on the recent experience of the peer.

Different models have been proposed for trust management in distributed networks [5,9]. [5] uses a global reputation management to evaluate distributed trust by aggregating votes from all peers in the network. Sun et al. [9] propose an entropy-based model and a probability-based model, which are used to calculate the indirect trust, propagation trust and multi-path trust. These models involve a lot of overhead and are not suitable for our system because IDSes can be easily compromised. They also suffer from collusion attacks.

Our model is also distinguished from the trust models developed for the application of e-marketplaces [10]. We introduce the concepts of expertise level and physical location to improve the accuracy of intrusion detection. We also allow IDSes to send test messages to establish better trust relationships with others. The alert risk ranking is categorized into multiple levels as well.

7 Conclusions and Future Work

In this paper, we presented a trust-based HIDS collaboration framework that enhances intrusion detection within a host-based IDN. The framework creates incentives for collaboration and we prove it is robust against common attacks on the collaborative network. The conducted simulations demonstrate the improved performance of our framework in detecting intrusions as well as its robustness against malicious attacks.

As future work, we will investigate the design of a communication protocol for the collaborative network, which takes privacy and efficiency issues into consideration. We will also design an automatic feedback to satisfaction level converting function, which takes risk levels of test messages, difficulty levels of test messages, and feedback from IDSes as inputs and generates satisfaction levels to the feedback as output.

Furthermore, we intend to extend our trust model to go beyond a generalized trust value for an HIDS. More specifically, since in practive HIDSes might have different expertise in detecting different types of intrusions, we would want to model the trustworthiness of a HIDS with respect to each individual type of intrusion. This will result in more effective trust management for assisting HIDSes to seek advice from truly helpful others. The subjectivity of HIDSes needs to be addressed when modeling their trustworthiness. HIDSes may have different subjective opinions on the risk levels of alerts. They can be more or less sensitive to certain intrusions.

Incentive design is another possible extension of our work. In the current protocol, the system may encounter free-rider problem such that some nodes forward the test messages to its neighbors and receive the rankings, then they forward the aggregated feedback from its neighbors to the tester. Free-riders can create unnecessary traffic in the network and degrade the inefficiency of the system. Honest users may be taken advantage and be deceived by dishonest “middle-agents”. In our future work, we will investigate this problem and create corresponding incentive design to discourage free-riders and reward honest participants.

Finally, we also intend to evaluate the resistance of our framework against collusion attacks; as well as investigate its scalability in terms of number of HIDSes, rate and type of attacks.

References

1. Aycock, J.: Painting the internet: A different kind of warhol worm. Technical Report, TR2006-834-27, University of Calgary (2006)
2. Douceur, J.: The sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429. Springer, Heidelberg (2002)
3. Duma, C., Karresand, M., Shahmehri, N., Caronni, G.: A trust-aware, p2p-based overlay for intrusion detection. In: DEXA Workshops, pp. 692–697 (2006)
4. Janakiraman, R., Zhang, M.: Indra: a peer-to-peer approach to network intrusion detection and prevention. In: WET ICE 2003. Proceedings of the 12th IEEE International Workshops on Enabling Technologies, pp. 226–231 (2003)
5. Jiang, T., Baras, J.: Trust evaluation in anarchy: A case study on autonomous networks. In: INFOCOM. IEEE, Los Alamitos (2006)
6. Li, Z., Chen, Y., Beach, A.: Towards scalable and robust distributed intrusion alert fusion with good load balancing. In: LSAD 2006: SIGCOMM workshop on Large-scale attack defense, pp. 115–122. ACM Press, New York (2006)
7. Moore, D., Shannon, C., Claffy, K.: Code-red: a case study on the spread and victims of an internet worm. In: IMW 2002: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, pp. 273–284. ACM, New York (2002)
8. Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Commun. ACM 43(12), 45–48 (2000)
9. Sun, Y., Han, Z., Yu, W., Liu, K.: A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks. In: INFOCOM. IEEE. Los Alamitos (2006)
10. Zhang, J., Cohen, R.: Trusting advice from other buyers in e-marketplaces: the problem of unfair ratings. In: ICEC 2006, pp. 225–234. ACM, New York (2006)

Multi-constraint Security Policies for Delegated Firewall Administration

Cássio Ditzel Kropiwiec[1], Edgard Jamhour[2], Manoel C. Penna[2], and Guy Pujolle[1]

[1] LIP6, UPMC, 104 Avenue du Président Kennedy
75016 Paris, France
[2] PPGIA, PUCPR, Rua Imaculada Conceição, 1155, 80215-901, Curitiba, Brazil

Abstract. This work presents a new policy based security framework that is able handle simultaneously and coherently mandatory, discretionary and security property policies. One important aspect of the proposed framework is that each dimension of the security policies can be managed independently, allowing people playing different roles in an organization to define security policies without violating a global security goal. The framework creates an abstract layer that permits to define security policies independently of how they will be enforced. For example, the mandatory and security property polices could be assigned to the risk management staff while the discretionary policies could be delegated among the several departments in the organization.

1 Introduction

In large networks, using a collection of firewalls increases the network security by separating public and private resources. It also permits to control the access of internal users to internal resources, reducing the risks of attacks originated from the inside of the network [1]. However, many difficulties arise when configuring large networks. First, it is necessary to determine the rule set that must be applied to each firewall, in a way that the overall security policy is satisfied [2]. Also, as firewalls of diverse models and vendors can be present, it is necessary to consider the specific set of rules that can be interpreted and enforced by each firewall in the network before applying the configuration. The topology of the network and the placement of the firewalls with respect to the users and resources is another aspect that must be considered. Each firewall receives a rule set according to its location in the network. If more than one firewall is present between a user and a resource, the rule set can be combined in order to better explore the distinct features offered by the firewalls. Ideally, the process of defining security policies should be decoupled from the mechanisms that will actually enforce them over the network. In most organizations, security policies are related to business goals, and are not anymore a purely technical issue.

In order to address the aforementioned issues, this paper proposes a policy-based security framework that introduces a new approach related to the security policy definition and the generation of firewall configuration in a distributed environment. The framework adopts a policy model with three dimensions of security policies:

mandatory, discretionary and security property. Mandatory policies are coarse grained and reflect the inviolable security restrictions in the organization. Discretionary policies are fine grained, and are subjected to the mandatory policies. While mandatory and discretionary policies are restrictions imposed to the right of access from users to resources, the security property policies are restrictions imposed to the paths connecting users to resources. In our framework a path must satisfy some security requirements in order to be allowed. This permits to create policies which are independent on the user or resource location. The motivation for this division is to support the cooperation of multiple security staff in the security policy definition. For example, the right to define mandatory and security property policies could be assigned to an organizational-level risk management staff while the discretionary policies could be delegated to the local administrators in several departments in the organization.

The process of translating the three-dimensional high level security policy into firewall configuration is highly complex. In order that the firewall configuration respects the high level definition, we have formalized both the policy model and the translating algorithm using the Z-language notation. The proof of some important theorems permits to demonstrate the coherence of our approach with respect of combining multiple policies.

The remaining of this paper is organized as follows: section 2 presents some representative related works. Section 3 describes our proposed framework, presenting both the security policy language and the translation algorithm. Section 4 presents the Z-language representation of the framework and the theorem proving. Section 5 presents a case study, illustrating how the framework works. Finally, Section 6 concludes the paper and points to future developments.

2 Related Work

Presently, it is possible to find numerous academic or commercial firewall languages proposed to simplify the firewall configuration process. These languages can be classified according to criteria such as vendor independence, topology independency and their level of abstraction.

Firewall languages are vendor dependent when they apply only to firewall devices of a specific vendor. Some examples are the Cisco PIX [3] e Cisco IOS [4] languages. On the other hand, vendor independent languages are not limited to a specific vendor. This is the case of INSPECT language patented by CheckPoint [5]. A vendor can create a firewall supporting the INSPECT standard by implementing a compiler that translates the INSPECT language into the firewall's native configuration instructions.

Languages are topology dependent when they were designed to represent the configuration of each firewall isolated, i.e., the placement of the users and resources with respect to the firewalls is taken into account by the network administrator and not by the language compiler. The language used by the framework presented in [6] is an example of topology dependent language. The framework represents firewall configuration as high level policies based on the Ponder specification [7]. Even though the high level language provides the use of symbols for masquerading host and network addresses, it is still topology dependent, because there is no automated strategy for selecting the sub-set of rules that applies for a specific firewall.

The languages are independent of topology when it is possible to represent both the security rules and the network topology independently. In this case, rules are not specific to each firewall. There is a mechanism or algorithm that evaluates the network topology (i.e., the placement of users and resources with respect to the firewalls) and translates the security policies into localized firewall configuration. The framework described in [8] is an example of topology independent firewall configuration. In [8] the access control policies are defined in three levels: organizational, global and local. Policies at the organizational level are described in natural language, and define security goals such as blocking offensive content and scanning actions. Organizational policies are transformed into global filtering rules at the global level. The subset of the global rules that concerns each firewall is separated and distributed at the local level.

The languages employed for firewall configuration can be further classified according to its level of abstraction. In low-level languages, the network configuration is represented by a set of rules of type “if conditions are satisfied than enforce actions”. The conditions are basically described in terms of the packet’s header fields. Most languages found in the literature, such as the one employed by the Firmato toolkit, are low-level [12]. On the other hand, the high-level languages uses a more abstract concept, wherein the security policies says “what must be done” instead of saying “how must be done”, i.e., the policy define an intention independently of the mechanisms used to implement it. Some examples of policy based languages can be found in [9], [10] and [11]. The framework presented in [9] permits to represent high-level policies in the form of a list of data access rules (DACL), that declares permissions of executing simple operations (read or write) on objects. The framework translates the high-level policies into low-level policies suitable to be configured into the firewall devices. An algorithm for checking the fidelity of the low-level policies with respect to the high-level policies is also presented. The project presented in [10] aims to automate the management of security policy in dynamic networks. The central component is a policy engine with templates of the network elements and services that validates the policy and generates the new security configurations for the network elements when the security is violated. The work presented in [11] abstracts hosts and area addresses by using names, which permits to easily determine which firewalls are traversed by the communication flows. It defines an algorithm that, given a specific topology, creates the filter set for each firewall or router. It also defines a second algorithm that verifies if the resulting configuration violates any of access policies.

The work described in [14] adopts a graphic representation of security rules. The work also defines the concept of security goals (e.g., top secret, mission critic, etc), which impose additional security properties that are required in order to access an object or perform a given access mode. At a lower level, a security goal is expressed in terms of a security requirement vector, which defines the minimum levels of properties such as confidentiality, integrity, availability and accountability. A security assumption vector defines the same properties assigned for the principal and elements along the path between the principal and the resource. In order of an access to be granted, it is necessary that all properties of the security assumption vector satisfy the corresponding properties in the security requirement vector. We have borrowed many concepts related to the security property model from this work.

3 The Framework

This work presents a new policy based security framework that is capable to handle simultaneously and coherently mandatory, discretionary and security property policies. The framework supports the definition of network security configuration for systems formed by a set of users willing to access a set of protected resources. A resource is a service delivered at a location, which can play the role of the source or the destination of an access. A source location is the place where a user initiates an access while a destination location is the place where one or more services are delivered.

A user can access a resource if there is permission. Permissions are represented by three different security models: mandatory, discretionary and security property. The mandatory model defines permissions by classifying users and resources with clearances and classifications. In the mandatory model, a permission is defined whenever a user classification is greater then a resource clearance [13]. The discretionary model defines permissions by mean of rules that relate users to resources, including their possible sources and destinations. Finally, the security property model defines permissions by assigning security levels to firewalls, and locations.

The framework has two main components: an information model and a refinement algorithm. The first includes all high level security information, whereas the second allows security policy formulated by high level statements to be consistently translated to firewall security rules.

3.1 The Information Model

The Information model is organized in five main blocks: the Inventory, the Mandatory Model, the Discretionary Model, the Security Property Model, and the Firewall Features Model, and is depicted in Figure 1.

The Inventory contains the objects and relationships necessary to build the security policy. It is organized in two main objects groups. The first includes users, locations, resources and services. A Service is modeled by the combination of a protocol and two port numbers for the source and destination sides. For instance, the Telnet service would be defined as TCP with destination port 23 and any source port. Although users interact with services, permissions are granted to resources. A Resource consists of one or more services delivered from one or more locations. For example, the E-mail resource could be defined to represent SMTP, POP and IMAP services. A Location is the place from where users access resources and also the place where a resource is located. Physically it corresponds to a host or subnet, and is represented by an IP address and a mask. When assigned to users, a location plays the source role, and the destination role when assigned to resources. The User Located At and Resource Located At classes indicate, respectively, the locations from where a user can initiate an access or from where a resource can deliver a service. Users, locations and resources can be organized in groups, what is modeled by a corresponding abstract class. For example, an Abstract User can be a User or a User Group, which in its turn can contains many abstract users, that is, many users or user groups.

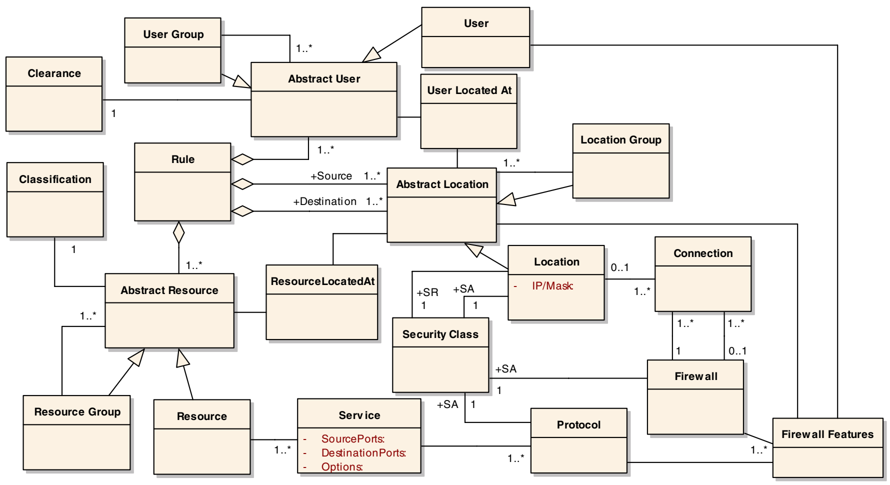


Fig. 1. The Information Model

The second object group includes connections, locations and firewalls. The Connection class represents connections between locations and firewalls or between two firewalls. Together, they model the network topology. The Firewall Feature class models firewall functionality, that is, its ability to perform some processing over the network packets.

The Mandatory Model establishes the mandatory access control policy by determining resource access according to a clearance versus classification schema. Clearance levels are assigned to users wherever classification levels are assigned to resources. A user is allowed to access a resource only if its clearance is equal to or greater than the corresponding resource classification.

The Discretionary Model is constructed by a set of discretionary rules that state the security actions to be enforced for specific service accesses. A service access includes the service, a user, and the location from where he can initiate an access; and a resource, and the destination from where it can be accessed. Examples of security actions are: accept, deny, log, and forward. In this study we just consider the accept action, and adopt the “anything not explicitly allowed is forbidden” strategy.

The Security Property Model provides fine-grained security information by including configuration and location dependent constraints. Security property rules enforce access control based on two properties, the security requirement (SR), which is defined for destination locations, and security assumption (SA), which is defined for source locations, firewalls and protocols.

Security requirements and security assumptions are specified by security levels within a security class. The security level is a natural number ranging from one to four, expressing the “strength” of one of the following security properties: confidentiality, integrity, availability and accountability. The security level establishes a total order over the security property set: the greater is the corresponding number (1, 2, 3 or 4) the stronger is security level. A security class is defined by an array of size four. If *sc* is a security class, *sc*[1], *sc*[2], *sc*[3] and *sc*[4] correspond, respectively, to its confidentiality, traceability, integrity and accountability security levels. Any entities that can be involved in resource accesses (i.e., locations, firewalls, and protocols) have a security class.

Security requirements and security assumptions can change when the related objects are combined. For example, John Doe trying to access a resource from the Engineering subnet would probably have a different SA than when he is trying to access the same resource from JD-Home host. Assuming that the corporation has much more control over the Engineering subnet then the first combination should result in a stronger SA. The protocol used by the object also changes its SA. For example, John Doe at Engineering subnet accessing a resource through HTTPS protocol introduces lower risk than when he is trying to access same resource from the same subnet through HTTP. Consequently a stronger SA should be assigned to the first. The modified security assumption is referred as effective security assumption (ESA). The upper effective class operation ($\cup$) over the security class set is defined to compute it. Let sc_1, sc_2, …, sc_n, to be security classes such that $sc_i = [x_{1i}, x_{2i}, x_{3i}, x_{4i}]$.

$$\bigcup_{i=1}^{n} sc_i = [\sup_i(x_{1i}), \sup_i(x_{2i}), \sup_i(x_{3i}), \sup_i(x_{4i})] \tag{1}$$

Security assumptions along an end-to-end path are combined together to form the overall security assumption (OSA). The permission to a user (from a source location) willing access a resource (at a destination location) is granted only if the end-to-end path OSA is at least as "strong" as the destination location SR. This involves the comparison of security classes. Because there is a partial order over the security class set, we are able to define its "strength". When sc_1 and sc_2 are security classes such that $sc_1 = [x_1, x_2, x_3, x_4]$ and $sc_2 = [y_1, y_2, y_3, y_4]$, the security class sc_2 is stronger than sc_1 if $y_i \geq x_i$, for $i = 1..4$.

The OSA is calculated as follows: First, compute the ESA for the (source location, protocol) pair and for each (firewall, protocol) pair along the path. Then, the resulting ESAs are combined along the end-to-end path. In this case the calculation should retain the set of weakest security levels. For this, the lower effective class operation ($\cap$) is defined over the security class set as follows. Let sc_1, sc_2, …, sc_n, to be security classes such that $sc_i = [x_{1i}, x_{2i}, x_{3i}, x_{4i}]$.

$$\bigcap_{i=1}^{n} sc_i = [\inf_i(x_{1i}), \inf_i(x_{2i}), \inf_i(x_{3i}), \inf_i(x_{4i})] \tag{2}$$

The Firewall Features Model contains the objects and relationships necessary to represent the firewall security functionality. The Firewall Feature class models the firewall ability to perform some processing over the network packets. For instance, a stateful firewall has the ability (or feature) of keeping track the state of network connections (such as TCP streams or UDP communication) flowing across it. This concept was introduced in Firmato [12] where the following features, included in the most common firewalls are listed: Names, Groups, IP Ranges, Stateful, Trust levels, Directional, Default Stance, Predefined Services and Layer.

A service access introduces the need for firewall features along the end-to-end path. For example, if the Email-plus resource represents a service that allows the exchange of e-mails with attached videos, then the Predefined Services feature must be present. The required features (RF) operation computes the set of all firewall features that are necessary for a service access. On the other hand, the feature can be supported by any of the firewalls along the path. In other words, the sequence of firewalls within a path supports the union of individual features. The virtual firewall (VF) operation computes the set of features supported along the path. If A = RF(service access) and B = VF(path), the service access is feasible if $A \subseteq B$.

A service access determines a firewall rule, that is, a sequence of conditions that must be satisfied to allow the access. We introduce the concept of firewall abstract rule, a vendor independent syntax firewall rule, that is, a sequence of abstract (vendor independent) conditions and the corresponding action. To build this abstract rule, one should consider the objects included in the service access, obtaining, for each one, the conditions registered in the inventory. For example, if John Doe is named in the service access and its login name in the inventory is JDoe, then the corresponding abstract condition is (login_name, JDoe). Also, if the Engineering subnet is named in the service access and its IP address is 10.1.1.0/24, the corresponding abstract condition is (IP_SRC, 10.1.1.0/24). Each abstract rule must be distributed along the firewalls involved in a service access. The features supported by each firewall are registered in the inventory. Please note that the RF operation assures that all necessary conditions can be enforced by the firewalls along the path.

3.2 The Refinement Algorithm

The refinement algorithm is presented in the following, supported by the example depicted in figure 2.

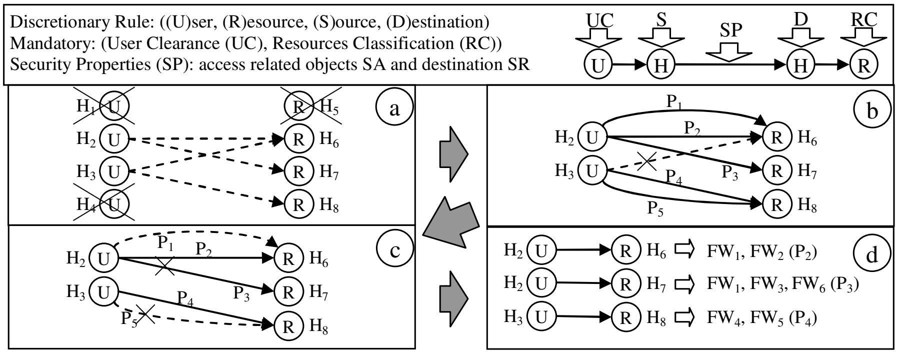

Discretionary Rule: ((U)ser, (R)esource, (S)ource, (D)estination)
Mandatory: (User Clearance (UC), Resources Classification (RC))
Security Properties (SP): access related objects SA and destination SR
UC
S
SP
D
RC
a
b
c
d
H₂ U → R H₆ ⇨ FW₁, FW₂ (P₂)
H₂ U → R H₇ ⇨ FW₁, FW₃, FW₆ (P₃)
H₃ U → R H₈ ⇨ FW₄, FW₅ (P₄)

Fig. 2. A Refinement Example

1. Identify the set of (user, resource) pairs that hold the discretionary and mandatory models: (i) Take all abstract users and resources that are referenced by discretionary rules. (ii) Expand groups to individual users and resources. (iii) Select all (user, resource) pairs for which user clearance is greater than or equal to resource classification.

2. For the (user, resource) pairs obtained in step 1, compute the set of service accesses, that is, ((user, source), (destination, resource), service) tuples, that hold the inventory and discretionary model (Figure 2a). The source and destination locations must be referred by a discretionary rule and be registered in User Located at or

Resource Located at, respectively. Each resource is related to one or more services. If the set is empty, terminate the algorithm.

3. Compute the set of possible paths for the service access computed in step 2 (Figure 2b): (i) For each service access, find a candidate path. (ii) Identify the possible protocols for the candidate path, from the services delivered at the corresponding destination. (iii) For each candidate path, compute the corresponding OSA. If it is at least as stronger as the destination SR, include it in the set of possible paths. (iv) If the set of possible paths is empty for all services accesses, terminate the algorithm.

4. Compute the set of feasible paths for the service accesses that corresponds to a possible path computed in step 3 (Figure 2c): (i) For each service access, compute A = RF(service access). (ii) For each path in the set of possible path, compute B = VF(path). (iii) If $A \subseteq B$, include the path in the set of feasible paths. (iv) If the set of feasible paths is empty for all services accesses, terminate the algorithm.

5. Compute the set of abstract firewall rules for each service access that corresponds to a feasible path computed in step 4 (Figure 2d): (i) For each firewall, compute the set of service access in which it is involved. (ii) For each service access in which the firewall is involved, produces an abstract rule with the conditions it can implement.

4 Formal Representation, Analysis and Validation

The refinement algorithm must guarantee that translation process doesn't cause the violation of the policies. High-level and low-level policies must represent the same set of permissions; otherwise, the whole system can be compromised. Two main theorems must be demonstrated to validate the algorithm:

1. Every access allowed by the higher level policy should be supported by the lower level policy (if they can be correctly enforced), and
2. No action allowed by the lower level policy should be forbidden by the higher level policy.

The formalism used in this work for formal validation and analysis is based on Z notation [15]. The Z notation is a formal specification language used for describing and modeling computing systems. The Z/EVES tool [16] is used to aid in representation and manipulation of Z notation. It is an interactive system for composing, checking, and analyzing Z specifications.

The validation approach used in this work consists of representing the algorithm in Z notation, creating theorems that represents the properties that the system must hold and using the Z-Eves tool to automatically prove these theorems, thus validating that the mathematical representation of the algorithm is consistent and complete.

The complete Z specification of the system and the proved theorems is very extensive and complex to be entirely presented in this paper. However, to illustrate how it is done, we present some excerpts in the following. The full Z specification is available to download at [17]. As an example, Figure 3 presents the procedure for handling discretionary rules.

```
DiscretionarySchema
1   $Rule: \mathbb{P}\ User \times \mathbb{P}\ Resource \times \mathbb{P}\ Location \times \mathbb{P}\ Location \times Action$
    ----------------
2   $Rule = (users, resources, sources, destinations, action)$
3   if $\exists user: users \bullet user = AnyUser$
4   then $RuleUsers = Users$
5   else if $users \subseteq Users$ then $RuleUsers = users$ else $RuleUsers = \emptyset$
6   if $\exists resource: resources \bullet resource = AnyResource$
7   then $RuleResources = Resources$
8   else if $resources \subseteq Resources$
9       then $RuleResources = resources$
10      else $RuleResources = \emptyset$
11  if $\exists source: sources \bullet source = AnySource$
12  then $RuleSourcesExplicit = Locations$
13  else if $sources \subseteq Locations$
14      then $RuleSourcesExplicit = sources$
15      else $RuleSourcesExplicit = \emptyset$
16  $RuleSourcesLocatedAt$
17    $= \{\ u: User; l: Location \mid u \in users \wedge (u, l) \in UserLocatedAt \bullet l\ \}$
18  $RuleSources = RuleSourcesExplicit \cap RuleSourcesLocatedAt$
19  if $\exists destination: destinations \bullet destination = AnyDestination$
20  then $RuleDestinationsExplicit = Locations$
21  else if $destinations \subseteq Locations$
22      then $RuleDestinationsExplicit = destinations$
23      else $RuleDestinationsExplicit = \emptyset$
24  $RuleDestinationsLocatedAt$
25    $= \{\ r: Resource; l: Location \mid (r, l) \in ResourceLocatedAt \bullet l\ \}$
26  $RuleDestinations = RuleDestinationsExplicit \cap RuleDestinationsLocatedAt$
```

Fig. 3. Line 2 defines the structure of the rule. Lines 3 to 5 select the users referenced by the rule in RuleUsers set. If AnyUser is present in the rule then the RuleUsers set must contain all the users registered in the system. Otherwise, the specification checks if the specified users are registered in the system and makes the RuleUsers set to include them if true. If the two previous verifications are false, then RuleUsers set is empty, meaning that the rule is not valid for any user. Lines 6 to 10 state the same logic for RuleResources set. Lines 11 to 18 specify how RuleSources set is built. Note that lines 11 to 15 are similar to RuleUsers set specification. The differences are at lines 16, 17 and 18. In the first two, the sources where the users can be located are selected, while in line 18, the RuleSources set is defined as the intersection between the sources specified in the rule and the locations of the users. The same logic is applied to RuleDestinations set at lines 19 to 26.

With the algorithm modeled in Z, the next step is to specify the theorems and to prove them. The two main theorems previously cited in this section were divided into several small theorems, in order to make the demonstration process simpler. We present some of these theorems in the following paragraphs.

Theorem 1. "If a rule specifies a user and a resource and if the user clearance is less than the resource classification, then the pair (user, resource) can not be a member of the *UsersAndResources* set". The Z-Eves code is presented in Figure 4.

Theorem 2. "For any protocol, source and destination allowed by the rule, if the OSA is smaller than SR, then the tuple *((protocol, source, destination), firewalls)* cannot be a member of *SecurePaths* set". The Z-Eves code is presented in Figure 5.

Theorem 3. "If the ((user, resource), (protocol, source, destination), firewalls) tuple defines a service access across the firewalls (i.e., if the tuple is a member of the *Permissions* set), then the rule must include these user, resource, source and destination; the user clearance must be equal to or greater than the resource classification; and the OSA for the *(protocol, source, firewall)* tuple must be equal to or greater than the destination SR". The Z-Eves code is presented in Figure 6.

theorem rule *testMandatory*
$MandatorySchema$
$\land\ user \in RuleUsers\ \land\ resource \in RuleResources$
$\land\ Clearance\ user < Classification\ resource$
$\Rightarrow \neg\ (user, resource) \in UsersAndResources$

Fig. 4. Z representation of Theorem 1

theorem rule *testSecurityProperties*
$SecurityPropertiesSchema$
$\land\ protocol \in Protocol\ \land\ source \in Location\ \land\ destination \in Location$
$\land\ firewalls \in \mathrm{seq}\ Firewall$
$\land\ ((protocol, source, destination), firewalls) \in RulePathsAndProtocols$
$\land\ \neg\ (OSA\ (protocol, source, firewalls), SR\ destination) \in GOSA$
$\Rightarrow \neg\ ((protocol, source, destination), firewalls) \in SecurePaths$

Fig. 5. Z representation of Theorem 2 - The *GOSA* set represents a relation between OSA and SR, and their elements are those for which OSA is equal to or greater than SR. Thus, the tuples (OSA, SR) that are not members of the GOSA set are those for which OSA is smaller than SR.

theorem rule *testPermissions*
$PermissionsSchema$
$\land\ ((user, resource), (protocol, source, destination), firewalls) \in Permissions$
$\Rightarrow user \in RuleUsers \land resource \in RuleResources$
$\land\ Clearance\ user \geqslant Classification\ resource$
$\land\ ((protocol, source, destination), firewalls) \in RulePathsAndProtocols$
$\land\ (OSA\ (protocol, source, firewalls), SR\ destination) \in GOSA$

Fig. 6. Z representation of theorem 3 - If some permission is a member of the *Permissions* set, then it must be present at discretionary, mandatory and security property policies

5 Example

According to our approach, only the framework has the credentials necessary to create rules in the firewalls. The policy administrators need to use the framework in order to manage the security policies. To illustrate the use of the framework, consider the example in Figure 7. It supposes an imaginary university network (yet realistic), with two firewalls separating 4 networks. The example illustrate how the mandatory and security property policies constraints the discretionary policies, avoiding violation of global security rules. For sake of simplicity, the security assumption and requirement vectors have been reduced to two dimensions: [confidentiality, traceability].

Suppose that the mandatory and security property policies have been previously defined (as presented in Figure 7) by a specialized department in the university, responsible for the overall security. Now suppose that an administrator responsible for creating discretionary policy decide to give full access permissions for all users with a discretionary rule such as: "*Any User from Any Location may Access Any Resource at Any Location*". In spite of this rule the discretionary rules generated by the framework would be defined as follows.

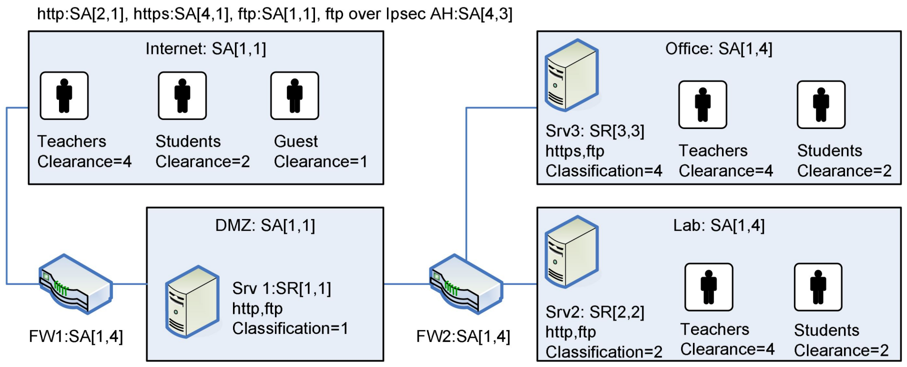


Fig. 7. Teachers and students can be located at Internet, Office or Lab network, and Guest users can only be located at Internet. Teachers have mandatory level (clearance 4) necessary to access resources http and ftp at Srv1 and Srv2, https and ftp at Srv3 (classifications 1, 2 and 3, respectively). Students have mandatory level (clearance 2) to access resources http and ftp at Srv1 and Srv2 (classifications 1 and 2, respectively). Guests users have mandatory level (clearance 1) to access resources http and ftp at Srv1 (classification 1). Considering the security property rules, https at Srv3 can be accessed from Office and Lab networks, and ftp can be accessed from any network if using IPsec AH. Srv2 can be accessed from Office network with http protocol. Srv1 can be accessed from any network (including Internet) using any protocol.

The rules for Firewall1 are:

- Permit Teachers, Students and Guests from Internet to access http and ftp services at Srv1 – because the clearance of these users are greater than the classification of resources at Srv1 and the OSA of the path [Internet, FW1] is equal to SR of Srv1.

- Permit Teachers from Internet to access ftp over IPsec AH at Srv3 – in this case, the OSA of the path [Internet, FW1, DMZ, FW2] combined with ftp over IPsec AH results in [4,3] that is greater than the SR [3,3] of Srv3 (the OSA were obtained from the combination of individual ESA of network elements, that for this situation are the following: Internet: [4,3], FW1: [4,4], DMZ: [4,3], FW2: [4,4]), but only Teachers have clearance (4) greater or equal to classification of resources at Srv3.

The rules for Firewall2 are:

- Permit that Teachers and Students access the http and ftp services at Srv1 from Office and Lab – both Teachers and Students have clearance greater than the classification of resources at Srv1, and the OSA of the combination these paths and protocols are greater than the SR [1,1] of Srv1.
- Permit Teachers and Students from Office to have access to http services at Srv2.
- Permit Teachers from Lab to have access to the https or ftp over IPsec AH services at Srv3.
- Permit Teachers from Internet to have access to the ftp over IPsec AH service at Srv3 – for the same reason of the second rule of Firewall1.

6 Conclusion

This paper has presented a framework capable of handling a multi-constraint security policy model. The security policy permits to create discretionary rules which are constrained by mandatory and security property policies. This is a very flexible approach that permits to describe a large number of discretionary rules without violating the primary security goals in a corporate environment. The motivation for this division is to support the cooperation of multiple security staff in the security policy definition. The policy model has been formalized and validated using the Z-notation and the Z-Eves tool. There are, however, many aspects to be considered in future studies. The methodology "anything not explicitly allowed is forbidden" should be replaced by a more flexible approach capable of supporting negative policies. Also, although a prototype has already been developed in Prolog, a broader scalability study is necessary.

References

1. Markham, T., Payne, C.: Security at the Network Edge: A Distributed Firewall Architecture. In: DARPA Information Survivability Conference and Exposition (DISCEX II 2001), vol. I, p. 279 (2001)
2. Al-Shaer, E., Hamed, H.: Discovery of Policy Anomalies in Distributed Firewalls. In: 23rd Conference of the IEEE Communications Society (INFOCOMM), pp. 2605–2616 (2004)
3. Cisco Systems Inc.: Cisco PIX Firewall Command Reference (2004), http://www.cisco.com
4. Cisco Systems Inc.: Cisco IOS Reference Guide (2004), http://www.cisco.com

5. CheckPoint Software Technologies Ltd.: Stateful Inspection Technology (2005), `http://www.checkpoint.com/products`
6. Lee, T.K., Yusuf, S., Luk, W., Sloman, M., Lupu, E., Dulay, N.: Compiling Policy Descriptions into Reconfigurable Firewall Processors. In: 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 39–48 (2003)
7. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Policy 2001: Workshop on Policies for Distributed Systems and Networks, pp. 18–39 (2001)
8. Haixin, D., Jianping, W., Xing, L.: Policy-Based Access Control Framework for Large Networks. In: Eighth IEEE International Conference on Networks, pp. 267–273 (2000)
9. Ou, X., Govindavajhala, S., Appel, A.W.: Network security management with high-level security policies. Technical report TR-714-04, Computer Science Dept, Princeton University (2004)
10. Burns, J., Cheng, A., Gurung, P., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A.V., Martin, J.D.M.: Automatic Management of Network Security Policy. In: DARPA Information Survivability Conference and Exposition, vol. II, pp. 12–26 (2001)
11. Guttman, J.D.: Filtering postures: local enforcement for global policies. In: IEEE Symposium on Security and Privacy, pp. 120–129 (1997)
12. Bartal, Y., Mayer, A.J., Nissin, K., Wool, A.: Firmato: A novel firewall management toolkit. ACM Transactions on Computer Systems 22(4), 381–420 (2004)
13. DOD: Trusted Computer Security Evaluation Criteria. DOD 5200.28-STD. Department of Defense (1985)
14. Albuquerque, J.P., Krumm, H., Geus, P.L.: Policy Modeling and Refinement for Network Security Systems. In: IEEE 6th International Workshop on Policies for Distributed Systems and Networks, pp. 24–33 (2005)
15. Spivey, J.M.: The Z notation: a reference manual. Prentice Hall International (UK) Ltd, Hertfordshire (1992)
16. Saaltink, M.: The Z/EVES system. In: Bowen, J.P., Hinchey, M.G., Till, D. (eds.) ZUM 1997. LNCS, vol. 1212, pp. 72–85. Springer, Heidelberg (1997)
17. Kropiwiec, C.D.: Z-specification for Firewall Policies, Algorithms and Theorem Proofs (2008), `http://www.ppgia.pucpr.br/jamhour/Research/`

Changes in the Web from 2000 to 2007

Ramin Sadre and Boudewijn R. Haverkort

University of Twente
Centre for Telematics and Information Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
{r.sadre, b.r.h.m.haverkort}@utwente.nl

Abstract. The World Wide Web has undergone major changes in recent years. The idea to see the Web as a platform for services instead of a one-way source of information has come along with a number of new applications, such as photo and video sharing portals and Wikis.

In this paper, we study how these changes affect the nature of the data distributed over the World Wide Web. To do so, we compare two data traces collected at the web proxy server of the RWTH Aachen. The first trace was recorded in 2000, the other more than seven years later in 2007. We show the major differences, and the similarities, between the two traces and compare our observations with other work. The results indicate that traditional proxy caching is no longer effective in typical university networks.

1 Introduction

Originally designed as a transport protocol for hypertexts, HTTP has become the leading "container" protocol for a large variety of applications in the last decade. Where in the past the web was primarily used for exchanging text-based information, possibly with some pictures, the web now is a major source of both texts and pictures of all sorts, but also of videos and music files. Yet, HTTP is being used for downloading most of these.

In 2000 we collected a trace from the RWTH proxy server, primarily to study object size distributions. Now, seven years later, we collected a similar trace, at the same "point" in the internet, and compared it, in many ways, with the 2000 trace. Apart from the object size distribution, we now also studied object types, and object constellation (sub-objects, links, and so on). In doing so, our aim has been to understand changes in the network traffic (volume as well as other characteristics) as generated by a large population of world-wide web users and how those changes affect the efficiency of the proxy cache.

We are not the first ones to study the characteristics of world-wide web traffic. Many studies have been reported so far (see the references at the end of this paper), however, not many do a comparison in time, as we do. Of the true comparable studies we found, we mention three. In [1], the authors study traces collected in 1999, 2001, and 2003 at the up-link of the University of North Carolina. Since they use TCP/IP headers only they do not examine the type of the documents requested by the clients. In [2], the accesses to three universitary servers are studied; instead we analyze the accesses of clients located in a university network to the whole WWW. Finally, [3] compares two traces collected at the Boston University in 1995 and 1998. It mainly focuses on the impact of the changes in the traces on different caching algorithms.

The remainder of the paper is structured as follows. In Section 2 we present the general characteristics of the two traces. In Section 3 we make a detailed comparison of various aspects of the traces: the response size and the type of the transferred documents (Section 3.1), the characteristics of the queried URLs (Section 3.2), the structure of the web pages (Section 3.3), and the cache efficiency of the proxy server (Section 3.4). We compare our major observations and conclusions with other work in Section 4. Finally, Section 5 summarizes the paper.

2 General Characteristics

The two traces that we examine in this paper have been extracted from the access log files of the Squid web proxy server [4] of the RWTH Aachen. The access log files have been collected at the proxy server from February 17, 2000 to March 12, 2000, respectively from August 6, 2007 to September 4, 2007. Both time periods are similar in the sense that they both fall into the semester breaks of the university. In 2000, main users of the university network were around 1950 scientists employed at the university and around 4350 students (out of 27400) living in student apartment blocks. These numbers have not changed much from 2000 to 2007.

For the following studies we have focused on the HTTP-GET requests that were processed with HTTP status code 200 (OK) or 304 (Not Modified) which were by far the most frequent status codes in the log files. In the following we denote the resulting data sets the "2000 trace", respectively the "2007 trace". We used parts of the 2000 trace in previous publications [5,6,7,8,9].

Table 1 shows some general characteristics of the two traces. Row 4 gives the overall number of requests recorded in the traces. Row 5 and 6 give the

Table 1. General characteristics

	2000 trace	2007 trace
start date	2000-02-17	2007-08-06
end date	2000-03-12	2007-09-04
#requests	26,318,853	18,747,109
#requests 200	20,734,319	13,621,849
#requests 304	5,584,534	5,125,260
#clients	2831	1124
#requests\ICP	19,716,054	16,415,251
#clients\ICP	2787	1119
#reqs/client\ICP	7074	14670
volume [Gbytes]	237	866

number of requests by status code. The (relatively) increased number of requests with return code 304 in the year 2007 indicates that client-side caching is now more common and efficient. Row 7 gives the number of unique clients that sent requests to the proxy server. These rows show that less queries have been sent by less clients to the server in the year 2007 than in the year 2000. It should be noted that not all clients represent single hosts since the proxy server is also queried by other proxy servers via the Internet Cache Protocol (ICP) [10]. When ignoring those other proxy servers, we obtain the numbers shown in rows 8 and 9. We observe that the number of clients decreased while the average number of requests per client (row 10) has increased by about 100% from 2000 to 2007. An explanation for the latter will be discussed in Section 3.3. The exact reason for the smaller number of clients is difficult to find because each department and student apartment block of the university is independently administrated and, hence, no information on the employed browser configurations is available.

Whereas the total number of requests has decreased, much more bandwidth has been consumed by responses with status code 200 in 2007 than in 2000. Row 11 shows the volume of transferred documents measured in Gbytes. The cause of this increase will be discussed in Section 3.1.

3 Detailed Comparison

3.1 Response Size and Type

In this section we study the size and type distribution of the responses with status code 200, i.e., responses that transferred an entire document to the client.

Response size distribution. Table 2 gives some important statistics of the response sizes (in bytes) as observed in the two traces. Although the median is similar for both traces, there are extraordinary differences between the two traces concerning the mean and the squared coefficient of variation of the response size distribution (SCV). In previous work [5,7,9], we observed that the response size distribution in the 2000 trace is heavy-tailed. The results shown in Table 2 suggest that the degree of heavy-tailedness has further increased over the last years. This is demonstrated by the log-log plots of the complementary cumulative distribution function (CCDF) of the response size shown in Figure 1.

Table 2. Response size statistics (in bytes)

	2000 trace	2007 trace
min	17	85
max	$0.228 \cdot 10^9$	$2.147 \cdot 10^9$
mean	12294.0	68275.2
median	2410	2780
SCV	320.9	3425.1

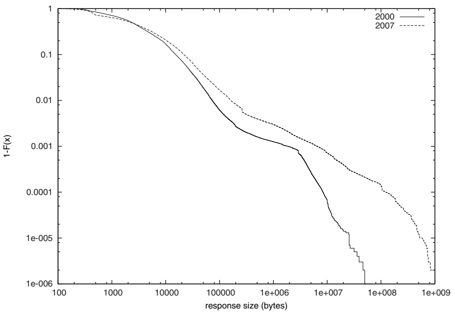


Fig. 1. CCDF of the response size

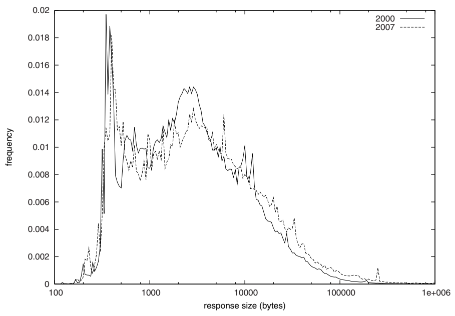


Fig. 2. Response size frequency (logarithmic bin size and normalized frequency)

We note that the CCDF of the 2007 trace decays significantly slower for very large response sizes than the CCDF of the 2000 trace, i.e., it is more heavy-tailed.

We are also interested in the changes of the distribution of smaller response sizes. Figure 2 shows for the two traces the histogram of the response sizes with logarithmic bin sizes and frequencies normalized to the trace size. As already expected from the increased degree of heavy-tailedness, we observe that the histogram of the more recent trace has a less developed waist in the range of 2000–5000 bytes, whereas the number of responses larger than 10000 bytes has considerably increased over a wide range. In addition, we observe that both histograms exhibit several distinct peaks. Explanations for these peaks can be found by a deeper analysis of the transferred documents in the following section.

Object types. To determine the types of the transferred documents we use the MIME type [11] information found in the Squid log files. Although the sent MIME type can be freely chosen by the server it is the only source of information for objects dynamically generated by server-side scripts or applications. We found 376 different MIME types in the 2000 trace and 384 different types in the 2007 trace. Table 3 show the frequencies of the ten most frequent types in the 2000 trace, respectively the 2007 trace. We observe a slight diversification of the types: In 2007, the five most frequent types only cover 88.2% of all requests, compared to 98.1% in 2000. Furthermore, it shows that the JPEG format and the PNG format have gained popularity over the GIF format. This could, however, be a consequence of improved client-side caching since GIF images are, in general, very small objects as shown in the following.

A different ranking is obtained when we examine the bandwidth consumed by the different types. Table 4 shows the percentage of the overall traffic volume (second column) consumed by the ten most bandwidth-consuming types in the 2000 trace, respectively the 2007 trace, and the corresponding average response size for documents of these types (third column). The last column gives a rough categorization of the content type. While in the year 2000, most of the traffic

Table 3. Number of requests by type for the 2000 trace (left) and the 2007 trace (right) as percentage of the trace size

type	#
image/gif	53.2%
image/jpeg	24.9%
text/html	18.4%
application/x-javascript	1.1%
text/plain	0.5%
text/css	0.5%
application/octet-stream	0.3%
image/pjpeg	0.2%
(unspecified)	0.1%
video/mpeg	0.1%

type	#
image/jpeg	33.3%
image/gif	28.5%
text/html	16.0%
application/x-javascript	6.9%
image/png	3.5%
text/css	2.5%
text/plain	1.8%
text/javascript	1.7%
application/octet-stream	1.3%
application/x-shockwave-flash	1.2%

Table 4. Fraction of traffic volume and average response size (in Kbytes) by type for the 2000 trace (left) and the 2007 trace (right) (i: image; t: text; v: video; a: audio; f: formatted).^{1} = application/octet-stream

type	volume	size	cat
image/jpeg	21.5%	10	i
image/gif	15.5%	4	i
text/html	14.6%	9	t
application/msword	9.0%	4147	f
application/octet-stream	8.4%	672	f
application/zip	8.1%	1322	f
video/mpeg	6.8%	861	v
application/vnd.ms-excel	2.5%	3637	f
text/plain	2.2%	49	t
audio/mpeg	2.1%	3360	a

type	volume	size	cat
application/octet-s.^{1}	34.6%	1766	v/f
image/jpeg	6.6%	13	i
application/x-otrkey	6.6%	240610	f
text/plain	6.1%	231	t
video/x-msvideo	6.0%	109533	v
video/x-flv	5.9%	10954	v
video/flv	5.4%	6730	v
video/x-ms-wmv	3.2%	42636	v
text/html	3.1%	13	t
application/zip	2.5%	9632	f

volume was caused by "traditional" web site components (i.e., HTML documents and images), we see that in the year 2007 nearly all traffic volume is generated by either video clips or archive-file downloads.[1] The average response sizes illustrate that documents of these types are responsible for the higher mean response size of the 2007 stream, as observed in Section 3.1. However, even the "traditional" types HTML and JPEG have increased in size by about 30%.

The response size distributions for the different document types help to understand the shape of the size distribution of the whole trace (see Section 3.1). In Figure 3, we show for both traces the response size histogram (with logarithmic bin size) for the whole trace (labeled "all'), for the most frequent document types ("images", "html" and, only for the 2007 trace, "x-javascript") and for all remaining types ("other"). Note that we have subsumed all image types (GIF,

[1] Note that the MIME type `application/octet-stream` is not very meaningful here. Only an inspection of the file name endings in the traces showed that most of the objects of that type are either video clips or software archives.

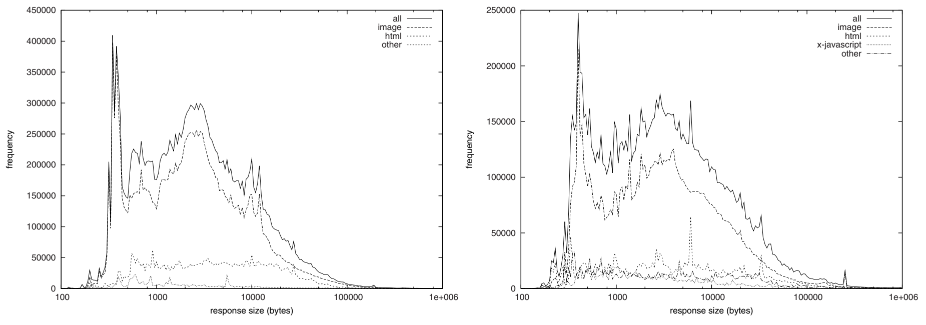


Fig. 3. Response size frequency by object type for the 2000 trace (left) and the 2007 trace (right) (with logarithmic bin size)

JPEG, PNG, etc.) under the “image” histogram. We do the following observations:

1. Image objects dominate the shape of the size distribution of both traces.
2. In the 2000 trace, the peaks in the range 300–400 bytes are caused by images from some few, but, at the time of the measurement, very popular web sites. 21% of all image requests in this range refer to only five servers (two of them are banner advertisment servers). In contrast, for the whole trace, the five most frequently queried servers are only accountable for 4.3% of all image requests.
3. In the 2007 trace, the three most distinct peaks are mainly caused by three sites. The peak at byte size 400 in the image histogram is created by about 100000 queries to `www.google-analytics.com`. In the peak at size 6200 in the HTML histogram, about 43000 queries refer to the download page of a file-sharing server. The Google search pages contribute with about 4000 queries to that peak. Similarly, the peak at the right end (byte size 250000) of the histogram nearly entirely consists of binary file downloads related to an online role-playing game.

3.2 Queried URLs

Some important characteristics of the queried URLs are summarized in Table 5. The second row gives the number of unique URLs queried by all clients. Row 3 gives the average number of requests per unique URL. Although the average did not change significantly from 2000 to 2007, the distribution of the number of requests per unique URL did, as illustrated by Figure 4. It shows, for both traces, the number of queries per URL normalized to the total number of queries. The shape of the log-log plot still follows Zipf’s law [12,3,13], but the slope has changed. As can be seen, the most popular URLs in the 2007 trace are much more often requested (in relation to the trace size) than their counterparts in the older trace. Only the most popular URL has a comparable request frequency in

Table 5. Characteristics of the queried URLs

	2000	2007
#URLs	6,846,724	4,896,006
#request/#URL	3.84	3.83
#URLs (client)	5282	6038
1x-URLs	65.1%	76.7%
#servers	137832	99147
#URLs/server	49.67	49.38

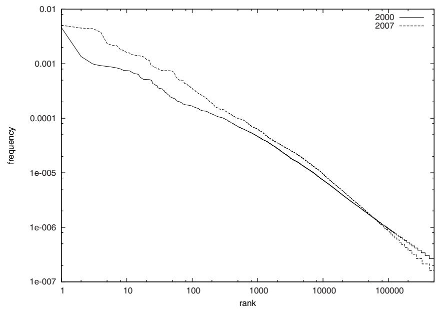


Fig. 4. Normalized number of queries per URL sorted by URL rank

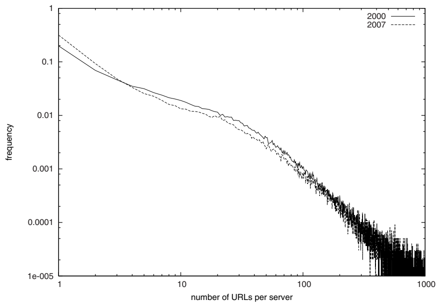


Fig. 5. Histogram of the number of URLs per server normalized to the number of servers

both traces. In addition, the average number of distinct URLs a client accesses has increased (row 4). Consequently, we observe an increase of URLs that have been requested only once during the whole measurement period. Row 5 shows their percentage in the two traces.

The last two rows show the number of queried servers (row 6) and the average number of unique URLs per server (row 7). Again, the average did not change while the distribution did as shown in Figure 5 which depicts for both traces the histogram of the number of URLs per server, normalized to the number of servers. We observe that the number of servers with 5 to 100 URLs has decreased. In contrast, the number of servers with only one or two distinct URLs has considerably increased. It is difficult to identify the reason for this change due the limited information stored in the traces. However, a manual inspection of the involved URLs showed that many of the "single-URL" servers are not "real" web sites but servers providing services to other web sites, such as visit counters, advertisements, etc. This implies that the interconnectivity of servers has increased: web servers offer less URLs but more often refer to other servers.

The servers with the largest number of distinct URLs are `www.geocities.com` in the year 2000 (98168 URLs) and `img.youtube.com` in the year 2007 (106675 URLs). However, these two servers are not the targets of the largest number

Table 6. Number of queries (as fraction of all queries) by server for the 2000 trace (left) and the 2007 trace (right)

server	queries
www.spiegel.de	1.02%
informer2.comdirect.de	0.98%
www.geocities.com	0.87%
www.heise.de	0.80%
www.africam.com	0.77%
ad.de.doubleclick.net	0.74%
www.ebay.de	0.68%
www.eplus.de	0.61%
www.consors.de	0.47%
ad.doubleclick.net	0.38%

server	queries
www.spiegel.de	2.03%
image.chosun.com	1.93%
www.svd.se	1.71%
img.youtube.com	0.85%
images.gmx.net	0.76%
www.bild.t-online.de	0.75%
pagead2.googlesyndication.com	0.68%
www.heise.de	0.67%
www.google-analytics.com	0.64%
www.manager.co.th	0.62%

of requests, as illustrated by the lists of the ten most queried servers shown in Table 6. We observe that in the year 2000 the five most queried servers received approximately the same number of requests, whereas in the year 2007 the difference between the servers at rank 1 and 5 was much larger.

3.3 Page Structure

An important question for the modeling of WWW traffic concerns the structure of web pages, that is, how many documents does a web browser have to fetch from the server in order to download the complete web page (called the *page size* in the following). This question is difficult to answer if proxy server traces are used as the only source of information. Given a sequence of queries sent by a specific client, the hardest problem is to identify the first and the last request for a specific web page.

A popular approach is based on the timestamps of the requests [1,14,15,16]. Let a and b be two consecutive requests sent by the same client at time s_a, respectively s_b, with $s_a < s_b$. The responses are sent back to the client at times e_a, respectively e_b. We define that two requests belong to the same web page if $s_b - e_a < \delta_{th}$ where δ_{th} is a predefined threshold. Note that $s_b - e_a$ can be negative if the client software sends requests in parallel. The best value for the threshold is usually unknown. It has to be larger than the time that the client software needs to process a response before it can send the next request. If the the threshold is too large, two requests belonging to two different web pages may be wrongly associated. In [1,14,15], a threshold of 1 second is used, assuming that the user certainly needs more than 1 second to react before calling the next web page.

We have calculated for both traces the mean page sizes that result from different thresholds (ignoring requests from other proxy servers, as described in Section 2). The results are shown in Figure 6. The figure illustrates that it is rather problematic to choose a specific threshold, such as 1 second, since the calculated mean page size continuously increases with the threshold. Nevertheless, we observe for a wide range of thresholds that the page size has doubled from

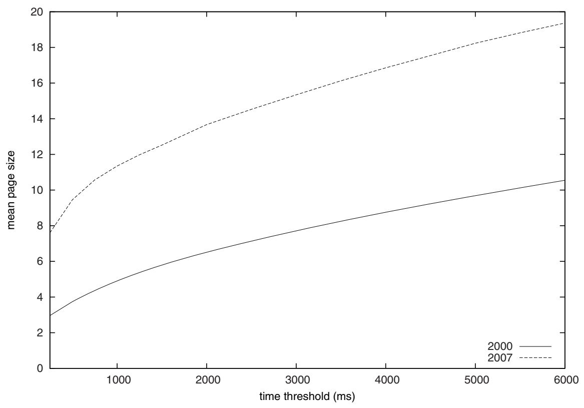


Fig. 6. Mean page sizes determined for different time thresholds for the 2000 and 2007 trace

Table 7. Cache efficiency and mean response size (in bytes)

trace	hit	miss	size hit	size miss
2000	54.3%	45.7%	8980	16230
2007	23.0%	77.0%	9846	85737

Table 8. Fraction of dynamically created responses by caching status

trace	total	hit	miss
2000	21.2%	1.2%	20.2%
2007	37.1%	4.3%	32.8%

the year 2000 to 2007, which explains very well the 100% raise of the number of requests per client as observed in Section 2. However, we advise to interpret these results with caution for the following reasons:

– Client-side cached documents may not be recorded by the traces.
– The user may call two web pages at the same time.
– Some requests in the trace do not necessarily reflect "normal" human behavior. For example, we have identified a series of 2300 requests in the 2000 trace with very small inter-request times. A user was obviously using a tool to download a complete copy of a web page. In another example, an advertisement banner caused a client to send over 10000 requests.

3.4 Cache Efficiency

Statistics about the caching behavior of the proxy server in 2000 and 2007 are shown in Table 7. The column titled "hit" and "miss" show the number of hits, respectively misses, as fraction of the total number of requests in the traces. The columns "size hit" and "size miss" give the mean size of the responses (in bytes) in case of a hit, respectively miss.

We observe a drop of the cache efficiency from 2000 to 2007. Whereas the mean size of cached documents has not changed very much, the probability for a cache hit significantly decreased. These numbers can not be directly compared because the proxy server has changed in terms of hardware and software from

2000 to 2007. However, we identify several reasons for the large number of cache misses that are more or less independent of the used server configuration:

1. The increased number of *very* large documents, as shown in Section 3.1. Since proxies only have limited cache memory such documents generate cache misses.
2. The increased number of URLs that have been only queried once, as discussed in Section 3.2.
3. The increased number of responses with status code 304 (see Section 2), indicating that client-side caching has improved its efficiency.
4. The increased number of dynamically generated responses, although that number can not be exactly determined because the traces do not include enough information. Especially the HTTP header lines controlling the cache behavior and the expiry dates would be helpful here.

An approach to identify dynamic documents is to scan the trace for URLs that frequently change their response size. This method, however, is not applicable here due to the large number of one-timer URLs. To get a very rough idea about the number of dynamic documents we have analyzed the URLs themselves. We have regarded an URL as dynamic when it contained form data or referred to scripts or server pages (file endings .php, .asp, etc.). Table 8 shows for both traces the resulting number of requests referring to dynamic documents as percentage of the number of all requests in the trace (column 2), of all requests causing a cache hit (column 3) and of all requests causing a cache miss (column 4). It shows that the number of requests to dynamic documents has increased and strongly contributes to the number of cache misses.

4 Comparison to Other Work

In the following, we give a summary of the important observations presented in the previous sections. For each observation, we give, if available, references to other publications that do (not) support it.

– Responses with status code 304 are more prevalent in 2007 than in 2000 (also reported in [2]).
– The volume of transferred documents (measured in bytes) significantly increased (also observed in [1,2,3]).
– The heavy-tailedness of the response size distribution increased from 2000 to 2007. The increase has been also reported by [1,2,3] but not at that extend. For example, the largest document observed in [2] for a trace from the year 2004 had a size of 193 Mbytes, whereas the largest file in our 2007 trace has a size of around 2 Gbytes. The fact that sizes are heavy-tailed distributed has been reported in many previous publications [17,18,19].
– "Traditional" web site components such as HTML documents and images are still the most frequently transferred objects. However, the most bandwidth consuming documents are now videos and compressed software archives.

Other authors have observed similar trends [2] but only for single web servers. We are not aware of recent studies on *proxy* servers of which the workloads, in our opinion, better reflect the overall trends in the Internet. The document type distributions found in the 2000 trace are consistent with the results reported in an older study on proxy server workloads [20].

- We have shown that a few popular servers can have a considerable impact on the size and type distribution of the transferred documents. The phenomenon of *concentration* [17,21], i.e., the fact that a large part of all requests concentrates on a few popular servers, has significantly increased (also reported by [2]). However, the popularity of the URLs still follows Zipf's law [2,3,12,13,19].
- The number of servers only hosting one or two URLs has considerably increased. Those servers are mostly providing services to other servers. This implies an increased inter-connectivity of web servers in the Internet, as also observed in [1].
- The number of one-timers (URLs only called once) has increased by 10%. In contrast, [2] reports a decrease by 50% for web servers. This illustrates the diversity of the documents as perceived by a proxy server in contrast to single web servers.
- The complexity of web pages (measured in number of URLs) has increased (confirmed by [1]). Older measurements can be found in [14,16].
- Cache efficiency for transferred documents has decreased. We believe that one reason is the significant amount of responses that are dynamically generated. Other publications report much lower amounts of dynamic documents [2,20].
- Our results indicate that traditional proxy servers that aim to cache *all* types of contents are not effective anymore in the studied network (and in networks with similar usage characteristics). However, specialized caching and distribution servers could provide better results as shown by simulation for YouTube videos in [22].

In this paper we have mostly focused on the stationary properties of the Web traffic. Some other publications have studied its time dynamics, especially correlations in the request rate [2], appearance of new documents [23], and the inter-reference time and the locality of references [2,24,25].

5 Summary

In this paper we have compared two data traces collected at the web proxy server of the RWTH Aachen. One was collected in 2000, the other seven years later in 2007. Through an elaborate data analysis, we show that the changes the World Wide Web has undergone in recent years have had a major impact on the volume and the nature of the observed traffic. Foremost, the size of the transferred documents has significantly increased due to the emerging popularity of bandwidth-consuming formats, such as videos and, surprisingly, binaries (software updates). At the same time, web pages have become more complex, i.e., they now consist of more objects, refer more often to other web servers, and more often rely on dynamically generated documents. For these (and some other) reasons, the efficiency of proxy servers has significantly decreased, indicating that traditional, non-specialized proxy servers that aim to cache all types of contents are not effective anymore. On the other hand, our measurement data also implies that client-side caching is now more common and efficient.

Our observations and their comparison to other work indicate that the changes that we have observed in our traces are, to some extend, representative for the global evolution of the World Wide Web. However, we realize that our conclusions are limited by the fact that only one server has been studied in this paper. Hence, we suggest that other researchers do similar comparisons with their old traffic measurements.

As for the future, we plan to continue our study of web traffic in order to observe changing workload patterns. At the same time, the traces we collected can be used for some other interesting studies, such as changes in user behavior and hyper-link topology.

Acknowledgments. We thank Wilfred Gasper, Rechenzentrum RWTH Aachen, for providing the anonymized 2007 trace.

References

1. Hernandez-Campos, F., Jeffay, K., Smith, F.: Tracking the Evolution of Web Traffic: 1995-2003. In: MASCOTS 2003, pp. 16–25 (2003)
2. Williams, A., Arlitt, M., Williamson, C., Barker, K.: Web workload characterization: Ten years later. In: Web Content Delivery, pp. 3–21. Springer, Heidelberg (2005)
3. Barford, P., Bestavros, A., Bradley, A., Crovella, M.: Changes in Web client access patterns: Characteristics and caching implications. World Wide Web 2(1-2), 15–28 (1999)
4. Squid (2007), http://www.squid-cache.org
5. El Abdouni Khayari, R., Sadre, R., Haverkort, B.: Fitting World-Wide Web request traces with the EM-algorithm. In: Proc. of SPIE 4523 (Internet Performance and Control of Network Systems), pp. 211–220 (2001)
6. El Abdouni Khayari, R., Sadre, R., Haverkort, B., Zoschke, N.: Weighted fair queueing scheduling for World Wide Web proxy servers. In: Proc. of SPIE 4865 (Internet Performance and Control of Network Systems III), pp. 120–131 (2002)
7. El Abdouni Khayari, R., Sadre, R., Haverkort, B.: Fitting World-Wide Web request traces with the EM-algorithm. Performance Evaluation 52(2-3), 175–191 (2003)
8. Haverkort, B., El Abdouni Khayari, R., Sadre, R.: A class-based least-recently used caching algorithm for World-Wide Web proxies. In: Kemper, P., Sanders, W.H. (eds.) TOOLS 2003. LNCS, vol. 2794, pp. 273–290. Springer, Heidelberg (2003)
9. Sadre, R., Haverkort, B.: Fitting Heavy-Tailed HTTP Traces with the New Stratified EM-algorithm. In: IT-NEWS 2008 - 4th International Telecommunication NEtworking Workshop on QoS Multiservice IP Networks, 2008 (QoS-IP), pp. 254–261 (2008)
10. Network Working Group: RFC 2186, Internet Cache Protocol (ICP), version 2 (1997), http://tools.ietf.org/html/rfc2186

11. Network Working Group: RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies (1996), http://tools.ietf.org/html/rfc2045
12. Zipf, G.: Relative frequency as a determinant of phonetic change. Reprinted from the Harvard Studies in Classical Philiology, Linguistic Society of America, vol. XL (1929)
13. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web Caching and Zipf-like Distributions: Evidence and Implications. In: INFOCOM 1999. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 126–134. IEEE Computer Society Press, Los Alamitos (1999)
14. Mah, B.: An Empirical Model of HTTP Network Traffic. In: INFOCOM 1997: Proceedings of the INFOCOM 1997. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 592–600. IEEE Computer Society, Los Alamitos (1997)
15. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. In: SIGMETRICS 1998/PERFORMANCE 1998: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, pp. 151–160 (1998)
16. Arlitt, M.: Characterizing web user sessions. ACM SIGMETRICS Performance Evaluation Review 28(2), 50–63 (2000)
17. Arlitt, M., Williamson, C.: Internet Web Servers: Workload Characterization and Performance Implications. IEEE/ACM Transactions on Networking 5(5), 631–645 (1997)
18. Crovella, M., Bestavros, A.: Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. IEEE/ACM Transactions on Networking 5(6), 835–846 (1997)
19. Crovella, M.: Performance Characteristics of the World Wide Web. In: Reiser, M., Haring, G., Lindemann, C. (eds.) Dagstuhl Seminar 1997. LNCS, vol. 1769, pp. 219–232. Springer, Heidelberg (2000)
20. Mahanti, A., Williamson, C., Eager, D.: Traffic analysis of a web proxy caching hierarchy. EEE Network, Special Issue on Web Performance 14(3), 16–23 (2000)
21. Arlitt, M., Jin, T.: A workload characterization study of the 1998 World Cup Web site. IEEE Network 14(3), 30–37 (2000)
22. Zink, M., Suh, K., Gu, Y., Kurose, J.: Watch global, cache local: Youtube network traffic at a campus network: measurements and implications. Multimedia Computing and Networking 2008 6818(1), 681805–681817 (2008)
23. Cherkasova, L., Karlsson, M.: Dynamics and Evolution of Web Sites: Analysis, Metrics and Design Issues. In: Sixth IEEE Symposium on Computers and Communications (ISCC 2001), pp. 64–71. IEEE Computer Society, Los Alamitos (2001)
24. Almeida, V., Bestavros, A., Crovella, M., de Oliveira, A.: Characterizing reference locality in the WWW. In: Fourth international conference on on Parallel and distributed information systems (DIS 1996), pp. 92–107. IEEE Computer Society, Los Alamitos (1996)
25. Arlitt, M., Friedrich, R., Jin, T.: Performance Evaluation of Web Proxy Cache Replacement Policies. In: Puigjaner, R., Savino, N.N., Serra, B. (eds.) TOOLS 1998. LNCS, vol. 1469, pp. 193–206. Springer, Heidelberg (1998)

Ensuring Collective Availability in Volatile Resource Pools Via Forecasting*

Artur Andrzejak[1], Derrick Kondo[2], and David P. Anderson[3]

[1] ZIB, Germany
andrzejak@zib.de
[2] INRIA, France
dkondo@imag.fr
[3] UC Berkeley, USA
davea@ssl.berkeley.edu

Abstract. Increasingly services are being deployed over large-scale computational and storage infrastructures. To meet ever-increasing computational demands and to reduce both hardware and system administration costs, these infrastructures have begun to include Internet resources distributed over enterprise and residential broadband networks. As these infrastructures increase in scale to hundreds of thousands to millions of resources, issues of resource availability and service reliability inevitably emerge. Our goal in this study is to determine and evaluate predictive methods that ensure the availability of a *collection* of resources. We gather real-world availability data from over 48,000 Internet hosts participating in the SETI@home project. With this trace data, we show how to reliably and efficiently predict that a collection of N hosts will be available for T time. The results indicate that by using replication it is feasible to deploy enterprise services or applications even on such volatile resource pools.

1 Introduction

Services are being deployed increasingly over large-scale computational and storage architectures. Internet services execute over enormous data warehouses (such as Google) and cloud computing systems (such as Amazon's EC2, S3). Scientific services are deployed over large-scale computing infrastructures (such as TeraGrid, EGEE).

To meet ever-increasing computational demands and to reduce both hardware and system administration costs, these infrastructures have begun to include Internet resources distributed over enterprise and residential broadband networks. Enterprises, such as France Telecom [1], are currently deploying a video-encoding service where the computational load is distributed among the peers in a residential broadband network. Universities, through academic projects such as

* This research work is carried out in part under the projects CoreGRID (Contract IST-2002-004265) and SELFMAN (contract 34084), both funded by the EC, and under the NSF award OCI-0721124.

Folding@home [2], deploy scientific applications that use PetaFLOPS of computing power from Internet resources.

Infrastructures of this scale are exposed to issues of availability and reliability. Low mean-time-to-failures (MTTF) and entire system crashes have plagued both service providers and customers. For example, when Amazon's S3 storage serviced crashed [3], numerous companies that depended on Amazon's services were stranded. In Internet environments, such as BOINC, resource unavailability can be as high as 50% [4].

At the same time, resource availability is critical for the reliability and responsiveness (low response latency) of services. Groups of available resources are often required to execute tightly-coupled distributed and parallel algorithms of services. Moreover, load spikes observed with Internet services (such as the commonly observed slashdot effect [5]) require guarantees that a collection of resources is available.

Given this need, our goal in this study is to determine and evaluate predictive methods that ensure the availability of a *collection* of resources. We strive to achieve collective availability from Internet distributed resources, arguably the most unreliable type of resource world-wide. Our predictive methods could work in coordination with a virtualization layer for masking resource unavailability.

Specifically, the contributions of this study are the following:

- We determine accurate methods and parameters for predicting resource availability. In particular, we investigate the factors that influence prediction error and determine indicators of resource predictability.
- We show how these methods and indicators can be used for predicting the availability of groups of resources and the associated costs. In particular, via trace-driven simulation, we evaluate our prediction method for collections of resources with different predictability levels. Our performance metrics include the success of predictions, and the cost in terms of redundancy and migration overhead for fault-tolerance.

Paper structure. In Section 2 we describe the data used in our study. Section 3 is devoted to the concept of predictability estimation, while Section 4 describes and evaluates the simulation approach used to ensure collective availability. Section 5 discusses related work. We conclude with Section 6.

2 Measurement Method

Our approach for gathering measurement data at a large-scale was to use the Berkeley Open Infrastructure for Network Computing (BOINC) [6]. BOINC serves as the underlying software infrastructure for projects such as SETI@home [7] and is deployed currently across over 1 million resources over the Internet.

We instrumented the BOINC client to record the start and stop times of CPU availability (independently of which application the BOINC local client scheduler chooses to run). The factors that can cause CPU unavailability include machines failures, power cycling, and user load. We term an *availability interval* as a period of uninterrupted availability delineated by a CPU start and the next CPU stop as recorded by the BOINC client. The BOINC client would start or stop an interval depending on whether the machine was idle. The meaning of *idle* is defined by the preferences of the BOINC client set by the user. We assume that the CPU is either 100% available or 0%. Our results in [4], and experience in [8] have shown this to be a good approximation of availability on real platforms.

This modified BOINC client was then made available for download starting on April 1, 2007. After a trace collection period of about seven months, the log files of these hosts were collected at the BOINC server for SETI@home on February 12, 2008. By December 1, 2007 more than 48,000 hosts had downloaded and were running the modified client. (By the end date, we had collected data from among 112,268 hosts, and the logs traced 16,293 years of CPU availability.) We use the subset of hosts (>48,000) that were actively running the client on December 1, 2007, and use the trace data for these hosts up to the end date of February 12, 2008.

Our trace data also includes the demographics of hosts, when specified by the BOINC user. About 32,000 hosts had specified host types. Of these hosts, about 81% are at home, 17% are at work, and 2% are at school.

3 Estimating Predictability and Forecasting Availability

This section focuses on forecasting the availability of individual hosts along with the estimation of the forecast accuracy. Our prediction methods are measurement-based, that is, given a set of availability traces called *training data* we create a predictive model of availability that is tested for accuracy with the subsequent (i.e. more recent) *test data*. For the sake of simplicity we refrain from periodic model updates.

An essential parameter for successful host selection is the expected accuracy of prediction (w.r.t. the test data). We designate it as *(host) predictability*. An essential ingredient of our approach is that we compute estimators of predictability from the training data *alone*. As we will show in Section 4, using this metric for host selection ensures higher availability ratios. Together with average availability, this metric allows also for fast elimination of hosts for which the predictions are less accurate and thus less useful.

3.1 Prediction Methodology and Setup

We compute for each host a predictive model implemented as a Naïve Bayes classifier [9]. A classification algorithm is usually the most suitable model type if inputs and outputs are discrete [10] and allows the incorporation of multiple inputs and arbitrary *features*, i.e., functions of data which expose better its information content. We have also tried other algorithms such as decision trees with comparable accuracy results. Since it is known that (given the same inputs and prior knowledge) no predictive method performs significantly better than others [11,12], we stick to this computationally efficient classifier.

Each sample in the training and test data corresponds to one hour and so it can be represented as a binary (01) string. Assuming that a prediction is computed at time T (i.e. it uses any data up to time T but not beyond it), we attempt to predict the complete availability versus (complete or partial) non-availability for the whole *prediction interval* $[T, T+p]$. The value of p is designated as the *prediction interval length* (*pil*) and takes values in whole hours (i.e. $1, 2, \ldots$). To quantify the prediction accuracy evaluated on a set S of such prediction intervals we use the ratio (called *prediction error*) of mispredicted intervals in S to $|S|$. The value of the parameter *pil* influences strongly the prediction accuracy. As other factors likely affect accuracy, we have also studied the length of the training data interval and the host type (i.e. deployed at home, work, or school).

To help a classifier, we enrich the original 01 data with features from the following groups. The *time* features include for each sample calendar information such as hour in day, hour in week, day in week, day in month etc. The *hist* features are (for each sample) the sums of the recent k "history bits" for $k = 2, 5, 10, 20, 50$ and 100. They express information about the length of the current availability status and the availability average over different periods.

3.2 Types of Predictable Patterns

There are several types of availability patterns that make a host predictable [13,14]. As a *type A* we understand behavior with long (as compared to *pil*) consecutive samples (stretches) of either availability or non-availability. Patterns of *type B* feature periodic or calendar effects, e.g. diurnal availability. For example, host availability over weekends or nights would create patterns of the later kind. Finally, availability of a host might occur after specific availability patterns (like alternating on/off status etc.). We disregard the last type as it is unlikely to occur in our scenario.

Knowing the predominant pattern type is helpful in designing the predictability indicators. For type A, simple metrics like the average length of an availability run might suffice, while type B requires more sophisticated methods like Fast Fourier Transformation (FFT). We identify the predominant patterns type by using different groups of features (from *hist* and *time*) in predictions and comparing the prediction accuracy. Obviously the *hist* features would lead to higher accuracy for type A patterns, while the *time* features are likely to be useful in presence of type B patterns.

3.3 Predictability Indicators

We have implemented as the predictability indicators a variety of metrics which are likely to indicate patterns of type A, B and possibly others. All indicators are computed over the training data only. The *aveAva* is the average host availability in the training data. The *aveAvaRun* (*aveNavaRun*) is the average length of a consecutive availability (non-availability) run. The *aveSwitches* indicator is the average number of changes of the availability status per week.

The last two indicators *zipPred* and *modelPred* are more involved and computationally costly. The former is inspired by [11] and is essentially the reciprocal value of the length of a file with the training data compressed by the Lempel-Ziv-Welch algorithm (raw, without the *time* and *hist* features). The rationale is that a random bit string is hardly compressible while a bit string with a lot of regularities is. To compute the last indicator, we train the classifier on half of the training data and compute the classification error (as above) on the other half. The *modelPred* value is then the reciprocal value of this error.

To compare indicators in their power to estimate the prediction error we compute the Spearman's rank correlation coefficient over all hosts for different *pil* values. We also verify the correlations by visual inspection of scatter plots (Figure 3).

3.4 Implementation and Running Time

The prediction and simulation experiments have been implemented and conducted using the Matlab 2007b environment. This framework was complemented by the PRTools4, a Matlab toolbox for pattern recognition [15]. As the Naïve Bayes classifier for predictions we used the `naivebc` method of PRTools4 with the default parameters.

Since Matlab is a partially interpreted language the running times of the algorithms are only upper bounds of tailored implementations. However, a complete experimental evaluation of a single host (including file operations, computation of *all* predictability indicators, classifier training and approximately 330 predictions in the test interval) required on average 0.25 seconds on a 2.0 GHz Xeon machine under Linux.

Even if this time is negligible compared to a typical *pil* value, an efficient, tailored implementation is likely to be much faster. Assuming that the length of the training interval in hours (i.e. the length of the 01 training string) is n, the upper bounds on the number of operations (per host) are as follows. The computation of *aveSwitches* is $O(n)$ with constant of 1 (other predictability indicators are not used in a production scenario). An efficient computation of the features *hist* and *time* requires time $O(n)$ with a constant below 10 in each case. An implementation of the Naïve Bayes classifier which precomputes all conditional probabilities during training requires time $O(nd)$ for training and $O(2d)$ for a prediction, where d is the number of features (below 20 in our case). As an example, assuming a training interval of 30 days ($n = 720$) and $d = 20$, a one-time feature computationa and training would require below $3 \cdot 10^4$ operations, and a single prediction about 40 operations. Note such a training is performed only if a host has passed the *aveSwitches* test which itself costs merely about $n = 720$ operations.

3.5 Experimental Evaluation

If not otherwise stated, the experiments used all the data described in Section 2 and the size of the training data was 30 days.

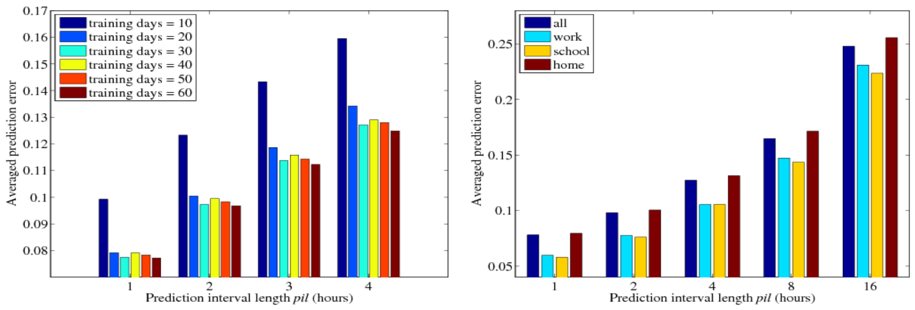


Fig. 1. Prediction error depending on the training data length and *pil* (left); Prediction error depending on the host type and *pil* (right)

Factors influencing the prediction error. Figure 1 (left) shows the dependence of the prediction error on the length of the training data and the *pil* value for a subset of 10,000 randomly selected hosts. While the error decreases significantly if the amount of training data increases from 10 to 20 days, further improvements are marginal. We have therefore used 30 days as the amount of training data for the remainder of this paper. Figure 1 (right) illustrates that the host type influences consistently the prediction error, with work and school hosts being more predictable. Despite of this effect, the simulation results did not show any significant differences between host types which can be attributed to low magnitude of differences.

Both figures show a strong influence of the prediction interval length, *pil*, on the prediction error. This is a consequence of increased uncertainty over longer prediction periods and the "asymmetric" definition of availability in the prediction interval (a short and likely random intermittent unavailability makes the whole interval unavailable).

Types of availability patterns. As described in Section 3.2 we measured the prediction error depending on groups of used data features for a group of

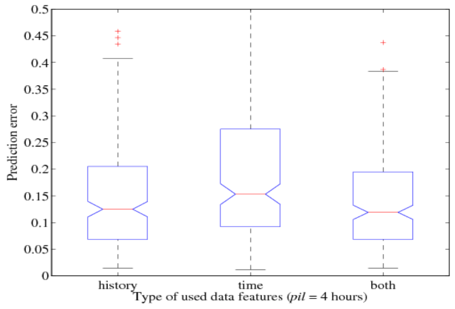


Fig. 2. Dependency of the prediction error on the data features

3000 randomly selected hosts. Figure 2 shows the lower quartile, median, and upper quartile values in the box while whiskers extend 1.5 times the interquartile range from the ends of the box. The median error is largest when using only *time* features and smallest when using both feature groups. While the magnitude of the difference is low, the relative order of cases was consistent across all *pil* values and host types. We conclude that most predictable hosts have availability patterns of type A (long availability state stretches) with few hosts exhibiting periodic or calendar predictability. Despite of this finding, we have included both the *time* and *hist* features in the classifier training as the computation of the them requires only linear time in the length of the training interval.

Evaluating predictability indicators. Table 1 shows correlations between prediction error and various predictability indicators defined in Section 3.3 (rows correspond to different prediction interval lengths). The strongest and most consistent correlation values has the *modelPred* indicator. However, as it is computationally most expensive, we identified the *aveSwitches* indicator as a viable substitute. For $pil = 1, 2, 4$ its correlation is comparable to *modelPred,* however it becomes much weaker for $pil = 8$ and 16. We could not fully explain this phenomenon, especially since the simulation results confirm its good quality even for these high *pil* values. However, a visual inspection of scatter plots for $pil = 2$ and $pil = 16$ (Figure 3) reveals that while for $pil = 2$ the correlation is obvious, for the higher *pil* value a relationship between the indicator and prediction error still exists but it is blurred by many "outliers". Finally, Table 1 confirms the finding that there is no clear relationship between average availability *aveAva* and the prediction error.

4 Evaluation of Group Availability Prediction Via Simulation

4.1 Method

We conducted trace-driven simulation applying the predictor determined in the previous sections. The predictor uses a training length of 30 days, which was shown to minimize prediction error according to Figure 1. This predictor is used to determine groups of available hosts, and the quality of the prediction is evaluated in simulation.

Table 1. Spearman's rank correlation coefficient between prediction error and various predictability indicators (rows correspond to different *pil* values)

pil	***aveAva***	***aveAvaRun***	***aveNavaRun***	***aveSwitches***	***zipPred***	***modelPred***
1	-0.370	-0.594	0.085	0.707	-0.654	-0.724
2	-0.275	-0.486	-0.011	0.678	-0.632	-0.690
4	-0.119	-0.303	-0.119	0.548	-0.502	-0.640
8	0.194	0.056	-0.245	0.195	-0.127	-0.642
16	0.211	0.091	-0.185	0.057	0.062	-0.568

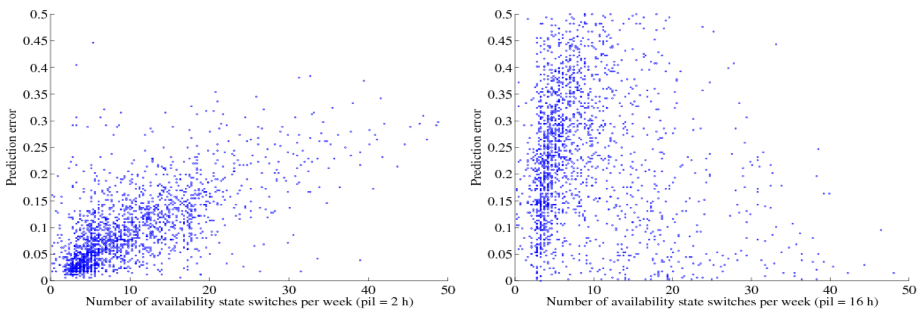


Fig. 3. Scatter plots of number of availability state changes per week (*aveSwitches*) and the prediction error (random subset of 2000 hosts)

We divide the hosts into groups based on the best predictability indicator described in Section 1, namely *aveSwitches*. To determine how to divide the hosts, we plotted the distribution of the *aveSwitches* values among the hosts (see Figure 4). The point (x, y) in Figure 4 means that y fraction of the hosts have an *aveSwitches* value of x or greater. Given the skew of this curve, we choose the median value of 7.47 near the "knee" of the curve to divide the hosts into two groups with high and low predictability respectively.

The parameters for simulation include the number of hosts desired to be available for some time period which is potentially longer than *pil*. In this case one could simply use the predictor at the end of each *pil* interval and repeat until the desired time period is reached. We refer to the total desired time period of availability as the *threshold* (which is some multiple of *pil*). For example, if the *pil* is 4 but the threshold is 12, we would rerun the predictor after time intervals of 4 and 8 elapse.

Another parameter is *redundancy*. We define redundancy to be $(R - N)/N$ where N is the number of hosts desired to be available, and R is the number of hosts actually used. The number of hosts we use in simulation are 1, 4, 16, 64, 256, 1024. The redundancy is in the range of [0, 0.50]. Due to space limitations, the simulation results we present below are for a *pil* of 4, though we also tried other *pil* values and observed similar trends.

The predictor is run using a test period of about two weeks which follow directly the training period. For each data point shown in the figures below, we ran about 30,000 simulations to ensure the statistical confidence of our results. In total, we executed more than 2 million simulations.

4.2 Performance Metrics

We measured the performance of the predictions in simulation by a *success rate* metric. In a simulation trial, we randomly choose R number of hosts from the pool predicted to be available for the entire threshold. We run trials in this way throughout the test period. We then count the number of trials where the number

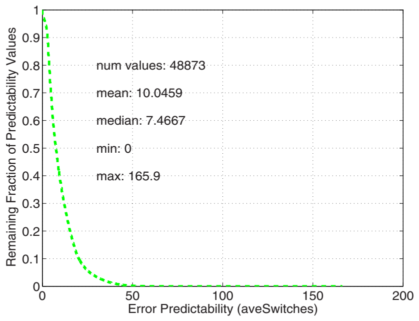


Fig. 4. Distribution of *aveSwitches*

of hosts actually available A (out of R) is greater than or equal to the desired number N. If A is greater than or equal to N for the entire threshold, then we consider the simulation trial a success. Otherwise, it is considered a failure. This fraction of the number of successes to total number of trials is defined as the success rate.

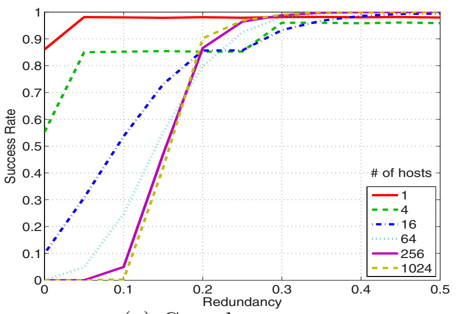
We also measure performance by the host *turnover rate* for thresholds greater in length than the *pil*. The turnover rate indicates the overheads (due to process migration, or service re-deployment, for example) due to changes in the predicted state of hosts from one sub-threshold to the next. We computed the turnover rate by first determining an active set of R hosts predicted to be available during some sub-threshold of length *pil*. In the following sub-threshold, we determine which hosts in the active set are predicted to be *unavailable*. The fraction of hosts in the active set that change from an available state to an unavailable state from one sub-threshold to the next is the turnover rate. Our method for computing the turnover rate gives an upper bound on fault-tolerance overheads.

We then compute the average turnover rate across all sub-thresholds of size *pil* throughout the entire test period. We conduct this process for active sets with different numbers of hosts and also different hosts randomly chosen from the pool of hosts predicted to be available.

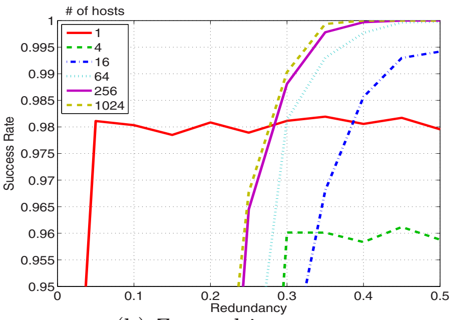
4.3 Results

In Figures 5 and 6, we show the success rate achieved for various levels of redundancy and number of hosts desired. Figure 5 focuses on the high predictability group, and Figure 6 focuses on the low predictability group. In each of the subfigures, we show the results in the entire range (left), and also for the zoomed-in range (right) for success rates in [0.95,1.00] or smaller.

In Figure 5 (left), we observe that when redundancy is 0, the success rate decreases exponentially with the number of hosts desired. However, as redundancy increases, the success rate increases dramatically as well. We observe the redundancy necessary to achieve success rates of 0.95 or higher in Figure 5 (right). In particular, if we look at the redundancy where the success rate is 0.95, we find that redundancy of 0.35 can achieve 0.95 success rates for groups up to 1024 in size.

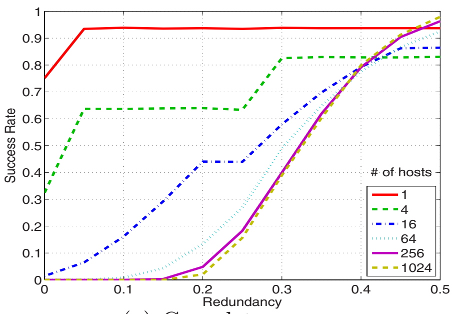


(a) Complete range

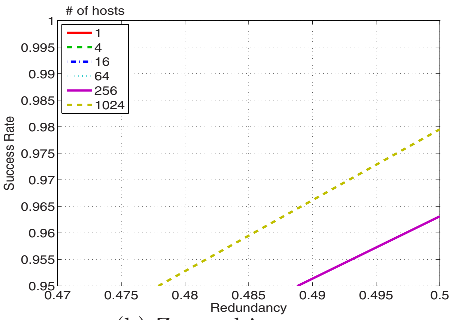


(b) Zoomed-in range

Fig. 5. Success rates versus redundancy for high predictability group and *pil* 4



(a) Complete range

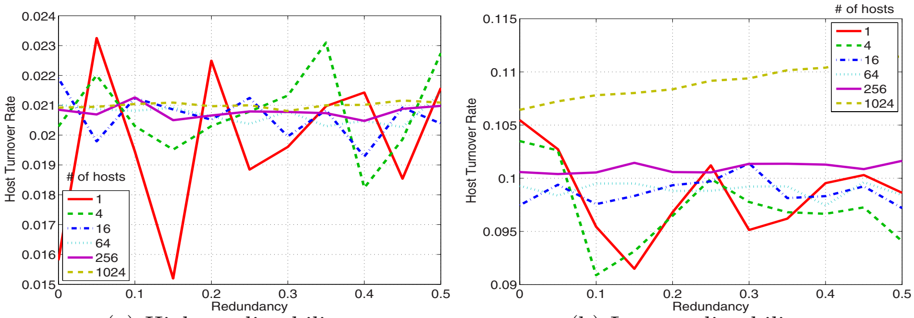


(b) Zoomed-in range

Fig. 6. Success rates versus redundancy for low predictability group and *pil* 4

In Figure 6, we observe similar trends in terms of success rate versus redundancy. However, we observe clear differences between the high and low predictability groups in the amount of redundancy needed to achieve the same level of group availability. For example, with the high predictability group, a redundancy of 0.35 will achieve success rates of 0.95 or higher. With the low predictability group, only the groups with 256 and 1024 desired hosts can achieve the same level of success rates; at the same time, high redundancy greater than 0.45 is required. Thus grouping hosts by predictability levels using the *aveSwitches* indicator significantly improves the quality of group availability prediction, and consequently the efficiency of service deployment.

Services will sometimes need durations of availability longer then the *pil*. In these cases, one can just reevaluate the prediction at the beginning of a new prediction interval, i.e., sub-threshold. The question is what are the costs in terms of service re-deployment across sub-thresholds. In Figure 7, we observe turnover rates as a function of redundancy and the number of hosts. We observe that the turnover rates are relatively low. The turnover rate is less than 0.024 for the high predictability group, and about 0.115 for the low predictability group.



(a) High predictability (b) Low predictability

Fig. 7. Turnover rates versus redundancy for *pil* 4

This shows that the overheads (of process migration, or using a checkpoint server, for example) with thresholds larger than the *pil* are relatively low. Observe that the stated turnover rates are for two consecutive *pil* intervals. The *cumulative* turnover rate for the whole threshold is $r * n$ where n is the number of sub-thresholds and r is the turnover rate.

Another important aspect of Figure 7 is the relationship between the number of hosts and turnover rate. We observe that the host turnover rate does not increase dramatically with the desired number of hosts. Instead, it increases only by a few percent even when the number of hosts increases by a factor of 4. This indicates that turnover rate scales with an increase in number of hosts. We also computed the turnover rates with a *pil* of 2. We found similar trends, and and the turnover rates were either equal or an order of magnitude lower.

5 Related Work

Forecasting is an established technique in the arsenal of proactive management of individual computer systems, with applications ranging from capacity estimation to failure prediction [10,12]. This study differs from other prediction studies in three main respects, namely types of hosts considered (inclusion of home desktops versus only those in the enterprise), the type of measurements used for the evaluation of prediction (CPU availability versus host availability), and the prediction issues investigated (the accuracy of efficient and scalable prediction methods, indicators of predictability, and the use and overheads of prediction of group availability).

With respect to the types of measurements, our data set consists of traces of CPU availability, which is a stricter and more accurate representation of resource's true availability. Most (P2P) availability studies [16,17,14] and prediction studies [14] focus on host availability, i.e., whether the host is pingable or not, instead of CPU availability. However, resources can clearly have 100% host availability but 0% CPU availability. Paradoxically, in our more volatile system, we find that simple prediction methods are suitable when applied with predictability indicators.

Moreover, with respect to the types of hosts characterized, the studies in [18,4] consider only resources in the enterprise (versus in home) environments. By contrast, the majority of resources in our study lie on residential broadband networks. Again, the time dynamics of CPU availability of home resources differ from those in enterprises. For example, the mean availability lengths found in this study are about 5.25 times greater than those in enterprise environments[4]. Also, the mean fraction of time that a host is available is about 1.3 times lower than that observed in enterprise desktop grids[4]. The studies in [19] and [20] focused solely on enterprise environments. For example, the Resource Prediction System (RPS) [19] is a toolkit for designing, building and evaluating forecast support in clusters. The Network Weather Service (NWS) is another well-known system for predicting availability in Grids (composed of multiple clusters).

With respect to prediction issues studied, we focus on novel prediction issues compared to previous works [14,19,20]. We focus on simple, scalable forecasting methods coupled with an efficient approach to filter out non-predictable hosts. Furthermore, we adjust our methods to types of predictable availability patterns for enabling group availability prediction.

6 Conclusions

In in this paper we attempted to show that a deployment of enterprise services in a pool of volatile resources is possible and incurs reasonable overheads. The specific technical contributions are this paper were as follows:

- We showed that the primary reason for the predictability of certain Internet hosts is the existence of long stretches of availability, and such patterns can be modeled efficiently with basic classification algorithms.
- We also demonstrated that simple and computationally cheap metrics are reliable indicators of predictability, and that resources could be divided into high and low predictability groups based on such indicators.
- For the high predictability group, via trace-driven simulation, we found that our prediction method can achieve 95% or greater success with collections of resources up to 1,024 in size using redundancy levels of 35% or less.
- For the high and low predictability groups, we found that the host turnover rates are less than 2.4% and 11.5% respectively. This indicates that prediction across long thresholds with low overheads is possible.

As a future work we plan to extend our experiments to other host pools, including PlanetLab. We also intend to study whether including additional features and inputs (such as CPU, memory or network utilization) can improve the prediction accuracy. Another research goal is to refine the predictability groups beyond low and high types.

References

1. Krishnaswamy, R.: Grid4all, `http://www.grid4all.eu`
2. Larson, S.M., Snow, C.D., Shirts, M., Pande, V.S.: Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. Computational Genomics (2003)
3. Carr, N.: Crash: Amazon's s3 utility goes down, `http://www.roughtype.com`
4. Kondo, D., et al.: Characterizing and Evaluating Desktop Grids: An Empirical Study. In: Proceedings of the IPDPS 2004 (April 2004)
5. Adler, S.: The slashdot effect: An analysis of three internet publications, `http://ldp.dvo.ru/LDP/LG/issue38/adler1.html`
6. Anderson, D.P.: BOINC: A system for public-resource computing and storage. In: Buyya, R. (ed.) Proceedings of 5th International Workshop on Grid Computing (GRID 2004), Pittsburgh, PA, USA, November 8, 2004, pp. 4–10. IEEE Computer Society, Los Alamitos (2004)
7. Sullivan, W.T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, G., Anderson, D.: A new major SETI project based on Project Serendip data and 100,000 personal computers. In: Proc. of the Fifth Intl. Conf. on Bioastronomy (1997)
8. Malecot, P., Kondo, D., Fedak, G.: Xtremlab: A system for characterizing internet desktop grids (abstract). In: Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing (2006)
9. John, G.H., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: Proc. 11th Conference on Uncertainty in Artificial Intelligence, pp. 338–345. Morgan Kaufmann, San Francisco (1995)
10. Vilalta, R., Apte, C.V., Hellerstein, J.L., Ma, S., Weiss, S.M.: Predictive algorithms in the management of computer systems. IBM Systems Journal 41(3), 461–474 (2002)
11. Keogh, E.J., Lonardi, S., Ratanamahatana, C.A.: Towards parameter-free data mining. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 206–215 (August 2004)
12. Andrzejak, A., Silva, L.: Using machine learning for non-intrusive modeling and prediction of software aging. In: IEEE/IFIP Network Operations & Management Symposium (NOMS 2008), April 7-11 (2008)
13. Douceur, J.R.: Is remote host availability governed by a universal law? SIGMETRICS Performance Evaluation Review 31(3), 25–29 (2003)
14. Mickens, J.W., Noble, B.D.: Exploiting availability prediction in distributed systems. In: NSDI, USENIX (2006)
15. van der Heijden, F., Duin, R.P.W., de Ridder, D., Tax, D.M.J.: Classification, Parameter Estimation and State Estimation. John Wiley & Sons, Chichester (2004)
16. Bhagwan, R., Savage, S., Voelker, G.: Understanding Availability. In: Proceedings of IPTPS 2003 (2003)
17. Saroiu, S., Gummadi, P., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proceedinsg of MMCN (January 2002)
18. Bolosky, W., Douceur, J., Ely, D., Theimer, M.: Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs. In: Proceedings of SIGMETRICS (2000)
19. Dinda, P.: A prediction-based real-time scheduling advisor. In: 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), vol. 10 (April 2002)
20. Wolski, R., Spring, N., Hayes, J.: Predicting the CPU Availability of Time-shared Unix Systems. In: Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8) (August 1999)

Adaptive Monitoring with Dynamic Differential Tracing-Based Diagnosis

Mohammad A. Munawar, Thomas Reidemeister, Miao Jiang, Allen George, and Paul A.S. Ward*

Shoshin Distributed Systems Group
University of Waterloo, Waterloo, Ontario N2L 3G1
{mamunawa, treideme, m4jiang, aageorge, pasward}@shoshin.uwaterloo.ca

Abstract. Ensuring high availability, adequate performance, and proper operation of enterprise software systems requires continuous monitoring. Today, most systems operate with minimal monitoring, typically based on service-level objectives (SLOs). Detailed metric-based monitoring is often too costly to use in production, while tracing is prohibitively expensive. Configuring monitoring when problems occur is a manual process.

In this paper we propose an alternative: Minimal monitoring with SLOs is used to detect errors. When an error is detected, detailed monitoring is automatically enabled to validate errors using invariant-correlation models. If validated, Application-Response-Measurement (ARM) tracing is dynamically activated on the faulty subsystem and a healthy peer to perform differential trace-data analysis and diagnosis.

Based on fault-injection experiments, we show that our system is effective; it correctly detected and validated errors caused by 14 out of 15 injected faults. Differential analysis of the trace data collected for 210 seconds allowed us to top-rank the faulty component in 80% of the cases. In the remaining cases the faulty component was ranked within the top-7 out of 81 components. We also demonstrate that the overhead of our system is low; given a false positive rate of one per hour, the overhead is less than 2.5%.

1 Introduction

Enterprise software systems are large and complex and their operators expect high availability and adequate performance. Proper operation given these conditions requires continuous monitoring. However, this increases operation costs since monitoring data is expensive to collect [1] and analyze [2]. System operators are faced with a choice of low-cost minimal monitoring, costly detailed monitoring, and prohibitively expensive tracing. Although detailed monitoring and tracing incur significant overhead, the information they provide is often essential for fault diagnosis. Given the performance ramifications, most enterprise software systems today operate with minimal monitoring, using a few key

* The authors gratefully acknowledge the support of IBM and the Natural Sciences and Engineering Research Council of Canada (NSERC).

metrics tied to service-level objectives (SLOs). When SLOs are violated a human operator enables detailed monitoring in an attempt to diagnose the fault, and, if unsuccessful, enables tracing. The quality of the final diagnosis is heavily dependent on operator skill, but they can err [3], and may be overwhelmed by the quantity of data collected.

We propose an alternative to this operator-driven approach: adaptive monitoring with dynamic tracing-based diagnosis. Our alternative has three steps: (1) error detection by monitoring of a minimal set of metrics *via* SLOs, (2) error verification by monitoring an extended set of metrics using invariant regression models, and (3) diagnosis using a differential analysis of ARM trace data collected from different peers. Each step is triggered based on analysis of data obtained in the previous step. Our approach is intended to keep the monitoring cost low during normal operation, only adding detailed monitoring and tracing when needed.

1.1 Background

Enterprise software systems comprise a mix of in-house, vendor-supplied, and third-party components, typically layered on standardized component frameworks like .NET [4], CORBA [5], J2EE [6], *etc.* To help operators monitor these systems, most components expose a variety of monitoring data at various granularity. Common data sources include performance metrics, correlated traces, and log records. In this paper we focus on J2EE-based systems, which provide monitoring data *via* management APIs such as Java Management eXtensions (JMX) [7] and ARM [8, 9]. Even though we focus on J2EE, the approach discussed here can be applied to other frameworks easily.

The JMX interface allows us to sample system metrics (*e.g.*, component response-time, activity count, resource pool status, *etc*) periodically. These are aggregate numerical values that reflect the state, behaviour, and performance of components. ARM traces contain fine-grained instance-level details such as order of component invocations and timing, which differ from the aggregate nature of metrics data. The order of component invocations can be combined to create complete paths. However, computing such paths in large-scale systems has a cost, especially when operations are logged out of order.

2 Approach

To illustrate our approach, we consider a simple J2EE cluster scenario, complete with load balancer, replicated application servers, and a common database back-end. We assume that the front-end load balancer can regulate the work assigned to a particular machine, and that admission control is in place to avoid system saturation. In addition, we assume that manifestation of multiple faults simultaneously in independent systems, is rare. As such, two faults, or even two instances of the same fault, are unlikely to be manifest in different application servers at the same time. However, a fault in a shared subsystem (*e.g.*, database) could affect all dependent subsystems.

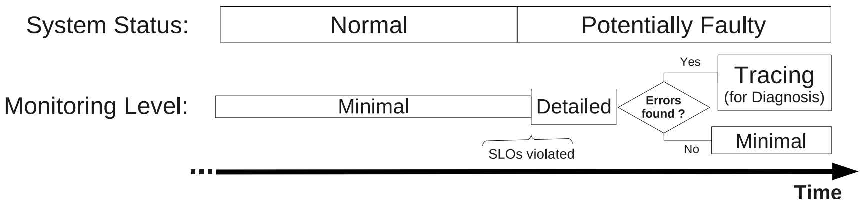


Fig. 1. Adaptive monitoring in action

Any monitoring system has to balance complexity, overhead, and error-detection capability. This insight guides our adaptive scheme: error hypotheses are generated with minimal cost; these are consequently verified and diagnosed within only a short period of performance degradation. Our approach has three steps: *Error detection with minimal monitoring*, *error verification with detailed monitoring*, and *trace-based diagnosis*. The effective cost of our system is then a function of the false-positive rate. Figure 1 depicts the operating steps of our approach, while Algorithm 1 describes its logic.

Minimal monitoring entails tracking a small set of important metrics and detecting errors using SLOs. Because of its low overhead, we use it when system conditions are deemed normal. Error verification involves monitoring a larger set of metrics, and is thus more costly. We use invariant relationships between metrics to track this larger set. When both SLOs and invariants indicate an error, we enable tracing on the suspected subsystem and one of its peers deemed healthy. We then perform precise diagnosis by comparing the collected traces. As we discuss in Section 3.1, detailed monitoring is costly, it is used only when

```
begin Monitoring
    mode := MINIMAL_MONITORING;
    while true do
        switch mode do
            case MINIMAL_MONITORING
                Monitor key metrics;
                if anomaly detected then
                    mode := DETAILED_MONITORING;
            case DETAILED_MONITORING
                Check invariant regression models;
                if error is confirmed for a cluster subsystem then
                    Reduce load submitted to the suspected faulty subsystem and a
                    non-faulty peer;
                    For both subsystems: mode := TRACING;
                if error is confirmed for all cluster members then
                    Report error in shared subsystems;
                if error is not confirmed then
                    mode := MINIMAL_MONITORING;
            case TRACING
                Compare trace data from the suspected faulty subsystem and a non-faulty
                peer;
                Diagnose based on the differences;
end
```

Algorithm 1. Pseudo-code of our approach

errors are detected in the first step. Tracing is costlier; it is enabled only when errors have both been detected with SLOs and validated with invariants.

2.1 Error Detection

Under normal conditions we only monitor a small set of carefully chosen metrics by comparing observed values against pre-specified SLOs in the form of thresholds. We check for SLO violations, which occur when the corresponding thresholds are consistently crossed. The monitored metrics are selected by system administrators based on four factors: they are sensitive to the state of a large subset of internal components, and so can detect a broad range of problems; they directly reflect users' perception of the service; they are inexpensive to measure and collect; and finally, problems that do not affect them are, by definition, not pressing enough to warrant further investigation. In our evaluation we monitor web page response times and number of failed requests of an Internet-based enterprise application. SLOs are typically defined based on contractual obligations (*i.e.*, service-level agreements) or user preferences. Alternatively, historical data can be used to set SLOs such that a certain percentile of observations are within specified bounds.

2.2 Error Verification

While SLO monitoring can be inexpensive, its simplicity and reliance on static thresholds make it vulnerable to false alarms. False alarms not only require administrators' time, they also increase the monitoring overhead, as tracing is unnecessarily triggered. The error verification step aims to limit the monitoring cost that arises because of false alarms, while providing a robust means for validating the existence of an error.

Our verification step entails collecting a larger set of system metrics, among which stable, long-term correlations exist [10, 11, 12, 13]. These correlations, also known as invariants, are captured *a priori* in the form of regression models using data collected from a healthy system. Each model associates two variables, one of which can be used to predict the other. As such, we can check each metric's behaviour by ensuring that its observed values are in line with predictions of the corresponding learned model. Verifying whether faults exist in the system involves determining the ratio of models that do not fit observations; when this ratio exceeds a specified level, the presence of faults is confirmed.

We have described our invariant-identification and error detection approach based on simple linear regression in previous work [12, 13]. Here we extend it to clustered systems, taking care to avoid identifying accidental correlations as invariants. Such correlations arise because of replication and coupling among subsystems. Replicated subsystems cause accidental correlations because they expose the same metrics and, because of the load balancer, experience similar workloads; as such, metrics that correlate within the subsystem, also correlate between replicas. We therefore only learn correlations within replicas, rather than between replicas. Similarly, coupling is caused by shared subsystems such as the load balancer and the database. We avoid these correlations by discarding models that relate metrics of any subsystem to metrics of a shared subsystem.

Error verification allows us to pinpoint faulty subsystems and identify healthy subsystems. We do so by analyzing invariants on each subsystem and then comparing the results across peers. A healthy subsystem will exhibit none or few model violations. If a single peer is experiencing model violations, we presume it is the faulty peer; where multiple peers are experiencing violations, per our assumption of a single faulty subsystem, we presume there is a common subsystem causing the errors. The outcome allows downstream diagnosis engines to readily select a non-faulty cluster member as a baseline.

2.3 Diagnosis

When an error is reliably detected, we enable ARM tracing on two peers: one healthy and one suspected to be faulty. The detection and verification steps determine which peer is deemed healthy and which is not; a non-healthy peer would cause SLOs and invariant correlations to be violated. At this stage, equal amounts of requests are directed towards the two peers, ensuring their behaviour will be statistically similar, modulo the fault.

Timing Data: Many faults directly or indirectly affect the timing behaviour of individual components. In particular, such faults change the distribution of the time taken to complete operations. To diagnose timing-related faults, we compare sample distributions of operation time instances, obtained from traces, of a healthy subsystem to those of the suspected non-healthy one. We use the standard χ^2 Two Sample Test [14] to check whether the two distributions of time values are different for a given significance level, α. This test does not require us to assume any specific timing distribution or handle sample-size differences. If the timing samples of a component are found to be statistically different, it is further considered in the diagnosis.

Our diagnosis consists of a ranked list of components. The rank of a component-operation pair is based on a score given by Equation 1, which is the ratio of the average execution time in the two different peers:

$$S(C_k) = \frac{\max(\mu_1(C_k), \mu_2(C_k)) + 1}{\min(\mu_1(C_k), \mu_2(C_k)) + 1} \tag{1}$$

$\mu_1(C_k)$ and $\mu_2(C_k)$ are the means of the two sample distributions for each component-operation C_k, and 1 is added to ensure numeric stability. We take the ratio of the maximum over the minimum value so that the full range of changes between the two peers is captured.

Structural Data: We also use weighted component-connectivity graphs derived from traces, represented as coincidence matrices, to diagnose faulty components. Each edge represents a caller-callee relationship and the edge weight is determined by the number of times the relationship appears in traces. We use the following three properties of the coincidence matrices for diagnosis:

1. InCalls: Calls made to component C_k, $\mathrm{IC}(C_k) = \sum_{i=1}^{m} G(i,k)$.
2. OutCalls: Calls made by C_k to other components $\mathrm{OC}(C_k) = \sum_{i=1}^{n} G(k,i)$.
3. OutCalls–InCalls Ratio: $\mathrm{CC}(C_k) = \frac{\mathrm{OC}(C_k)}{\mathrm{IC}(C_k)}$.

We use the same anomaly scoring function as for timing data, *i.e.*, Equation 1 is applied to the structural measures $\mathrm{IC}(C_k)$, $\mathrm{OC}(C_k)$, and $\mathrm{CC}(C_k)$, to produce individual scores and rankings. Because we aggregate the structural data in the form of counts, we ignore all counts smaller than the minimal threshold t_{min}.

2.4 Diagnosis Integration

Each of the diagnosis methods focuses on distinct characteristics of system behaviour. We therefore combine their results to improve fault coverage. Let $\mathbb{C}$ be the set of all components, and $\mathbb{M}$ be the set of diagnosis methods. For each $c \in \mathbb{C}$, a method $m \in \mathbb{M}$ reports an anomaly score $s = m(c) \in [0, \infty)$. $m(c) = 0$ if m does not shortlist c as anomalous. The goal of integration is to combine individual scores, $m_i(c)$, into a global score, $\hat{s}$:

$$\hat{s}:(m_1(c), m_2(c), \ldots, m_k(c)) \rightarrow [0, \infty] \quad \forall c \in \mathbb{C} \quad m_1, m_2, \ldots, m_k \in \mathbb{M}$$

Simple Combination: We normalize the anomaly scores of the individual methods, m_i, to the range $[0, 1]$ and take their sum as the combined anomaly score $\hat{s}$.

$$\hat{s} = \sum_{i=1}^{k} m_i'(c) \; \forall c \in \mathbb{C} \qquad m_i'(c) : m_i(c) \rightarrow [0,1], i = 1, 2, ...k \; \forall c \in \mathbb{C}$$

Weighted Combination: Some diagnosis methods can more accurately identify certain faults than others; this can be automatically determined *via* the diagnosis results. For example, faults that affect operation times can clearly be diagnosed by methods based on timing data; *i.e.*, there is a large gap in anomaly score between the faulty and non-faulty components. We thus give such methods more weight when their results indicate that they are accurate. Specifically, we weight the scores produced by the analysis of timing data by the ratio of the scores of the first- and second-ranked component. The other methods are assigned weights as described in the simple combination approach.

3 Evaluation

We set up a small clustered enterprise application environment to evaluate our adaptive-monitoring approach and differential trace-analysis methods. Our testbed, shown in Figure 2, comprises a DB2 UDB 8.2 database server, two WebSphere 6 Application Servers (WAS), workload generators, a monitoring engine, and a fault-injection module. We use this infrastructure to execute the Trade benchmark [15], a J2EE application which implements an online stock-brokerage system.

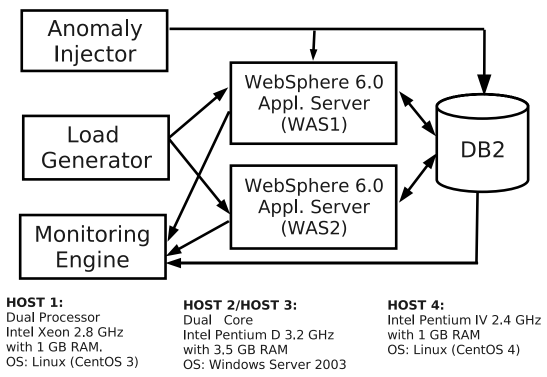

Anomaly Injector
Load Generator
Monitoring Engine
WebSphere 6.0 Appl. Server (WAS1)
WebSphere 6.0 Appl. Server (WAS2)
DB2
HOST 1: Dual Processor Intel Xeon 2.8 GHz with 1 GB RAM. OS: Linux (CentOS 3)
HOST 2/HOST 3: Dual Core Intel Pentium D 3.2 GHz with 3.5 GB RAM OS: Windows Server 2003
HOST 4: Intel Pentium IV 2.4 GHz with 1 GB RAM OS: Linux (CentOS 4)

Fig. 2. Experimental Setup

The monitoring engine collects ARM and JMX data from the two application servers. The metric data is collected every 10 seconds from the JMX interface of the application servers and saved in a local database. For convenience, unless otherwise stated, the metric data is collected throughout experiments for offline analysis. Trace data is logged in files only when enabled; the files are then fetched by the monitoring engine for analysis. While we could transfer the trace data directly from ARM agents to the monitoring engine, we did not see a performance gain when compared to logging.

The load generator creates randomized workload patterns, subject to a maximum. Because we simulate user activity, we implemented the load-balancing logic as part of the load generation module. In practise, a separate workload balancer would distribute the work among cluster members. When tracing is enabled, the load-balancing logic reduces and controls the load such that a roughly equal amount of work is submitted to each application server.

3.1 Cost of Monitoring

We performed experiments to measure the overhead caused by the monitoring logic (*e.g.*, counter updates and time-stamping) and the data-collection logic. We used a simplified setup with one application server, a database, and an open-loop workload generator that enforces exponential inter-arrival time between requests. The service time in similar systems has been observed to follow the exponential distribution [16]. We thus model the system using an M/M/1 queue and derive the mean service time (T_s) thus:

$$T_s = \frac{1}{\mu} = \frac{T_r}{\lambda T_r + 1} \tag{2}$$

where μ is the mean service rate, T_r is the response time, and λ is the request arrival rate. For each monitoring configuration, we execute experiments at different load levels. We repeat each experiment five times at every load level. For

Table 1. Effect of monitoring on service time

	Service Time		
Monitoring Level	*Mean (ms)*	*Std. Error (ms)*	*Overhead (%)*
None	7.468	0.001	0.0
Minimal	7.604	0.001	1.8
Detailed	10.152	0.009	35.9
Logged Trace	12.414	0.089	66.2

each experiment, our analysis takes 60 samples (10 minutes) into consideration; it excludes data collected during warm-up. Table 1 shows the mean service time as a function of increasing monitoring levels. The service times shown represent averages across the different load levels and the five repetitions. The standard error reflects the variation in the mean results. These results confirm that minimal monitoring has a small effect on performance, whereas detailed monitoring and tracing significantly degrade performance. While this increase may be acceptable for a brief period, it cannot be incurred continually. In particular, the overhead translates directly into the additional fraction of machines required for a data center with monitoring to service an equal load as one without, not taking into account analysis machines. Tracing has higher overhead than detailed monitoring. It also generates a larger amount of monitoring data which needs to be analyzed. Our analysis does not account for such overhead. These results support our claim that to contain the performance impact, detailed monitoring should only be used when an error is suspected, and that tracing should only be enabled if the error is confirmed.

Given these overhead numbers, we can estimate the cost of our adaptive monitoring scheme by assuming a false positive rate of less than one SLO triggered per hour, which is much more lenient than any system administrator would accept. With our setup and uniformly random workload, we have found that, using percentile-based SLOs determined from historical data, such a false alarm target can easily be achieved, while maintaining good fault coverage. Since detailed monitoring can refute the false positive in a minute, the mean service time of our approach would be $\frac{59}{60}7.604 + \frac{1}{60}10.152 = 7.646$, or an overhead of 2.39%.

Beside the measurement overhead, the analysis needed for our three-step approach incurs little computational overhead. During minimal monitoring, 20 metrics are tracked and each SLO can be checked in constant time. During detailed monitoring, the cost of analysis is $O(m \cdot i)$ where m is the number of invariant models and i is the number of sampling intervals considered. In our experiments

Table 2. Faults Parameters

Parameter	*Value*	*Parameter*	*Value*
d_{len}	500 (ms)	d_{wait}	$[0, 100)$ (ms)
d_{max}	∞	e_{prob}	0.3
l_{interval}	1000 (ms)	l_{lock}	0.5

m was close to 20000 for both application servers and detailed monitoring was enabled for 6 sampling intervals. The analysis needed for tracing depends on the method used. The cost of computing the InCalls and OutCalls measures is $O(c^2 \cdot s)$ where c is the number of components and s is number of entries per component in the traces. For analyzing the timing data, we need $O(c \cdot s)$ operations. In our experiments c, the number of components, was 81.

3.2 Faults and Fault-Injection Experiments

We have developed several types of faults to assess the effectiveness of our approach and methods. *Delay-loop faults* entail delaying completion of a selected method for d_{len} time units. To configure these faults, we specify a component, one of its methods, the delay-loop duration d_{len}, and a maximum number of loop instances d_{max}. These faults can be configured in two ways: (1) uniform-randomly spaced, by specifying a minimum random wait time between two delay-loop-executions d_{wait}; (2) probabilistic occurrences, by setting a probability of occurrence e_{prob}. *Exception faults*, with probability e_{prob}, throw an unhandled exception when a selected method is executed. A *table-locking fault* periodically locks a chosen database table. The lock is activated for l_{lock} fraction of every l_{interval} time interval during the fault-injection period. We have also developed a series of common operator-caused configuration faults, including authentication errors, incorrect thread-pool and connection-pool sizing, and component deletion. We do not evaluate these faults in this work because of space limits.

Each of our fault-injection experiments consists of a warm-up period, a period of normal activity during which we learn invariant regression models, and a period during which the system is monitored. We inject faults in the last period, while our monitoring is active. Unless stated otherwise, we use values listed in Table 2 to configure faults. The typical duration of an experiment is one hour.

3.3 Error Detection

During minimal monitoring, we oversee response-time and failure count metrics associated with all web pages of the Trade application. If either requests to a page fail or the response time of a page violates the corresponding SLO in three consecutive sampling intervals, we suspect presence of faults. Table 3 shows the effectiveness of our error detection approach. Each row represents an experiment where faults are injected in a method of a chosen component. We inject delay-loops and exception faults in components of WAS1. Table-locking faults are injected in the database.

The results show that all faults injected in WAS1 are detected. In the case of `QuoteBean`, we see SLO violations on the non-faulty application server (WAS2), which arise from the coupling induced by the database. We explain this further in Section 3.4. For table-locking faults, we see that, except for `ORDEREJB`, SLOs are violated on both WAS1 and WAS2. This is the expected behaviour, as both depend on the database.

Table 3. Error detection with SLOs and verification with invariant-correlation models

Faults		Error Detection		Error Verification	
Type	Component	Detected on *WAS1*?	Detected on *WAS2*?	% Model with Outliers *WAS1*	% Model with Outliers *WAS2*
Exceptions	QuoteBean	Yes	No	4.8	0
	OrderBean	Yes	No	5.6	0
	HoldingBean	Yes	No	4.8	0
	AccountProfileBean	Yes	No	0.5	0
	AccountBean	Yes	No	1.7	0
Delay Loops	QuoteBean	Yes	Yes	2.1	0.01
	OrderBean	Yes	No	3.3	0
	HoldingBean	Yes	No	2.9	0
	AccountProfileBean	Yes	No	1.1	0
	AccountBean	Yes	No	2.7	0
Table Locking	QUOTEEJB	Yes	Yes	2.1	1.7
	HOLDINGEJB	Yes	Yes	0.6	1.3
	ORDEREJB	No	No	0.3	0.4
	ACCOUNTEJB	Yes	Yes	2	1.9
	ACCOUNTPROFILEEJB	Yes	Yes	1.8	1.5

3.4 Error Verification

In this work we only leverage intra-subsystem invariant correlations for each cluster member (*i.e.*, intra-WAS in our case). An example of such a correlation is shown in Figure 3 where the number of requests to the `TradeScenario` component is plotted against the number of store operation for the `OrderEJB` component for both WAS1 and WAS2.

The intra-WAS models allow us to perform diagnosis at the level of cluster members. A subsystem is faulty if a significant number of its invariants are violated. In practise, the administrator would decide what is significant based

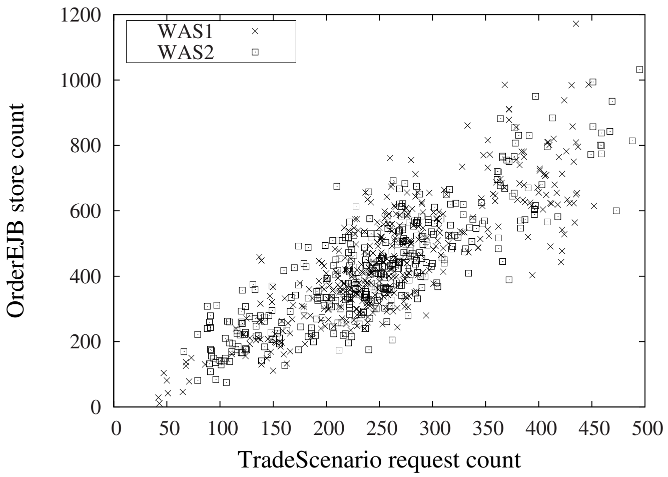


Fig. 3. Example of a stable correlation in both WAS1 and WAS2

on data collected during validation of the invariants. Because the target system is dynamic with many sources of noise and the invariant models are statistical, we cannot expect all models to hold at all times. Thus, after a thorough validation, we expect none or a very small fraction of models to report outliers. Here, we consider faults to exist if 0.5% or more of intra-WAS models report outliers.

Table 3 summarizes results obtained from the analysis of correlation models for the experiments reported in Section 3.3. For both loop and exception-based faults, at least 0.5% of models within WAS1, the faulty application server, report outliers. Except for `QuoteBean` with loop-based faults, no model from WAS2 persistently reports outliers. Loop-based faults cause database connections to be tied up for longer periods and also cause the associated threads on the database to be unavailable. This limits the number of threads available to process work from WAS2. This problem is visible in the case of `QuoteBean`, as it is the most frequently used component of Trade. Nevertheless, very few models report outliers on WAS2 in this case. We are thus able to correctly confirm an error in WAS1 in all cases.

Results for table-locking faults show a significant number of models reporting outliers on both WAS1 and WAS2 for all cases, indicating an error in the database. Tracing is thus not enabled, as it will not provide additional information on the status of the application servers. For the `ORDEREJB` table, the fraction of models with outliers is below the 0.5% threshold. In Table 3, we see that these faults are not detected using SLOs, thus neither detailed monitoring nor tracing is enabled. This case represents a false negative for our monitoring system.

3.5 Diagnosis

We now evaluate the accuracy of diagnosis methods and their combination. Table 4 summarizes the results obtained with the different methods. The first 10 results in Table 4 correspond to the first 10 results in Table 3; these represent experiments in which we apply all three steps, *i.e.*, detection, verification, and diagnosis if required. Note that diagnosis using the χ^2 test on the timing data only reports components whose operation time distribution changes at the 0.05 significance level.

We see two interesting phemomena in these results: (1) no one method is better than the rest, as the effectiveness of individual methods depends on the fault and the faulty component; (2) combining diagnoses can produce a better outcome than any single test; (3) the weighted combination generally beats the simple combination, but is sometimes (about 20% of the cases) worse than the best individual test.

For delay-loop faults, as expected, ranks using timing data are generally correct, while results with structural data are misleading. Given this, the weighted combination produces an exact diagnosis for the first set of experiments with delay-loop faults (lines 6–10), and only becomes weaker when the frequency of timing faults drops very low, to about 3% of the time (lines 16–20).

We perform additional experiments to evaluate the sensitivity of diagnosis based on trace data. The last 10 lines in Table 4 show the results of these

Table 4. Ranks of faulty (component, operation) out of 81 possibilities for delay loop- and exception faults. A rank of 1 is the most accurate diagnosis, while a rank of ∞ means that the component is not short-listed.

Fault Type	Faulty Component	Timing Data (OT)	OutCalls/ InCalls (CC)	InCalls (IC)	OutCalls (OC)	Simple Combination	Weighted Combination
Exceptions	QuoteBean.getDataBean	1	∞	1	∞	2	1
	OrderBean.getDataBean	9	2	1	3	1	1
	HoldingBean.getDataBean	7	2	6	2	1	1
	AccountProfileBean.getDataBean	1	∞	42	∞	9	6
	AccountBean.getDataBean	6	1	68	9	6	7
Delay loops	QuoteBean.getDataBean	1	∞	35	∞	7	1
$d_{\text{len}} = 500$ms	OrderBean.getDataBean	1	5	11	8	1	1
$d_{\text{wait}} = (0, 100)$ms	HoldingBean.getDataBean	1	∞	61	28	8	1
	AccountProfileBean.getDataBean	1	∞	39	∞	5	1
	AccountBean.getDataBean	1	4	20	13	1	1
Delay loops	QuoteBean.getDataBean	1	∞	48	∞	7	1
$d_{\text{len}} = 500$ms	OrderBean.getDataBean	1	4	32	13	2	1
$d_{\text{wait}} = (0, 1500)$ms	HoldingBean.getDataBean	1	∞	34	12	6	1
	AccountProfileBean.getDataBean	1	∞	16	∞	5	1
	AccountBean.getDataBean	1	3	13	8	1	1
Delay loops	QuoteBean.getDataBean	1	∞	20	∞	6	2
$d_{\text{len}} = 100$ms	OrderBean.getDataBean	1	4	33	20	3	1
$d_{\text{wait}} = (0, 7000)$ms	HoldingBean.getDataBean	3	∞	12	12	6	7
	AccountProfileBean.getDataBean	1	∞	11	∞	4	1
	AccountBean.getDataBean	1	4	32	16	2	1

experiments. We show that we are able to exactly diagnose delay-loop faults with loops lasting 500 ms with a random inter-loop gap with range $(0, 1500)$ ms, and within the top-7 even with a small disturbance of 100 ms with a random inter-loop gap with range of $(0, 7000)$ ms.

4 Related Work

Work in the area of error detection and diagnosis in enterprise software systems can be broadly divided into issues of data acquisition and analysis. Much work in data acquisition focuses on reducing monitoring overhead. Agarwala *et al.* [17] propose classes of channels, each with different rate and granularity of monitoring data; consumers can dynamically subscribe to these channels as needed. This approach only allows control of the communication overhead. Recent work on dynamic code instrumentation (*e.g.*, [18, 19]) focuses on how to efficiently adapt monitoring logic but not on when to do so.

Comparative analysis based on peer subsystems has been applied to offline diagnosis of known configuration faults (*e.g.*, [20, 21]). Pertet *et al.* [22] apply peer analysis to group communication protocols. But while their work presumes that the effects of the fault have spread, we assume efficient detection and validation, allowing timely and precise diagnosis. Kiciman and Fox [23] use peer comparison of paths to identify application-level faults. They, however, require continuous trace collection. Mirgorodskiy *et al.* [24] describe a methodology to compare timing behaviour of similar processes in a parallel computing environment. We focus on enterprise software systems which are often more dynamic than applications targeted in [24].

The use of invariant correlations between metrics for error detection and diagnosis was proposed both in our previous work [10] as well as by Jiang *et al.* [11]. The latter work assumes that a fixed set of metrics is always collected and no adaptation occurs. Agarwal *et al.* [25] also describe an approach to create fault signatures based on correlation between change-points in different metrics. Our prior work [12] is the first to demonstrate automated adaptive monitoring, and focuses on achieving the benefits of continuous monitoring at a fraction of the cost. The current work augments our earlier approach by diagnosing faulty components using more-precise trace data instead of metric-based invariants. Furthermore, the context of this work is a larger, clustered system, which allows us to employ novel techniques such as differential trace analysis.

Trace data analysis has been studied extensively. Kiciman and Fox [23] perform statistical comparison of instance-level component interactions. By contrast, our trace analysis is more efficient as it uses aggregate component interactions, and also looks at timing data. Kiciman and Fox also apply decision trees to correlate failures with faulty components. Likewise, Chen *et al.* [26] use clustering and Cohen *et al.* [27] use Bayesian models for the same purpose. Our work differs from these works in that they ignore monitoring costs.

5 Conclusions

In this paper we describe our approach to adaptively monitor software systems. Our approach consists of three, increasingly costly, steps: detection, verification, and diagnosis. To the best of our knowledge, it is the first monitoring system that automatically adapts from SLOs to tracing. It is also the first work that uses peer comparison of invariant models based on metrics for error detection. Unlike prior work that assumes continuous tracing, we enable detailed monitoring and tracing only when needed, thus incurring less than 2.5% overhead. Our verification approach uses peer comparison of invariant models of system metrics. We devise diagnosis methods based on differential analysis of information extracted from ARM traces and describe means to integrate their results. We show that once a statistically significant problem is detected, it is accurately validated and diagnosed. Our approach ensures that we only enable costly tracing when we are confident that tracing will accurately diagnose the defect.

References

1. Fox, A., Patterson, D.: Self-repairing computers. Scientific American (June 2003)
2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
3. Pertet, S., Narasimhan, P.: Causes of failure in web applications. Technical Report CMU-PDL-05-109, CMU Parallel Data Lab (December 2005)
4. Microsoft Corp: NET Platform, `http://www.microsoft.com/net/`

5. Object Management Group, Inc.: Common object request broker architecture (CORBA), `http://www.corba.org/`
6. Sun Microsystems, Inc.: Java 2 platform enterprise edition, v 1.4 API specification, `http://java.sun.com/j2ee/1.4/docs/api/`
7. Sun Microsystems Inc.: JMX - Java Management Extensions, `http://java.sun.com/javase/technologies/core/mntrmgmt/javamanagement/`
8. Johnson, M.W.: Monitoring and diagnosing application response time with ARM. In: SMW (1998)
9. Rolia, J., Vetland, V.: Correlating resource demand information with arm data for application services. In: WOSP (1998)
10. Munawar, M.A., Ward, P.A.: Adaptive monitoring in enterprise software systems. In: SysML (June 2006)
11. Jiang, G., Chen, H., Yoshihira, K.: Discovering likely invariants of distributed transaction systems for autonomic system management. In: ICAC (2006)
12. Munawar, M.A., Ward, P.A.S.: Leveraging many simple statistical models to adaptively monitor software systems. In: Stojmenovic, I., Thulasiram, R.K., Yang, L.T., Jia, W., Guo, M., de Mello, R.F. (eds.) ISPA 2007. LNCS, vol. 4742, pp. 457–470. Springer, Heidelberg (2007)
13. Munawar, M.A., Ward, P.A.: A comparative study of pairwise regression techniques for problem determination. In: CASCON, pp. 152–166 (2007)
14. Croarkin, C., Tobias, P. (eds.): Engineering Statistics Handbook, National Institute of Standards and Technology (2006)
15. Coleman, J., Lau, T.: Set up and run a Trade6 benchmark with DB2 UDB. IBM developerWorks, `http://www128.ibm.com/developerworks/edu/dm-dw-dm-0506lau.html?S_TACT=105AGX11&S_CMP=LIB`
16. Tesauro, G., Das, R., Jong, N.K.: Online performance management using hybrid reinforcement learning. In: Proceedings of SysML (2006)
17. Agarwala, S., Chen, Y., Milojicic, D., Schwan, K.: QMON: QoS- and utility-aware monitoring in enterprise systems. In: ICAC (2006)
18. Dmitriev, M.: Profiling java applications using code hotswapping and dynamic call graph revelation. In: WOSP, pp. 139–150 (2004)
19. Mirgorodskiy, A.V., Miller, B.P.: Autonomous analysis of interactive systems with self-propelled instrumentation. In: MMCN (January 2005)
20. Mickens, J., Szummer, M., Narayanan, D.: Snitch: Interactive decision trees for troubleshooting misconfigurations. In: SysML (April 2007)
21. Wang, H.J., Platt, J.C., Chen, Y., Zhang, R., Wang, Y.M.: Automatic misconfiguration troubleshooting with peerpressure. In: OSDI, p. 17 (2004)
22. Pertet, S., Gandhi, R., Narasimhan, P.: Fingerpointing correlated failures in replicated systems. In: SysML (April 2007)
23. Kiciman, E., Armando, F.: Detecting application-level failures in component-based internet services. IEEE Trans. on Neural Networks 16(5), 1027–1041 (2005)
24. Mirgorodskiy, A.V., Maruyama, N., Miller, B.P.: Problem diagnosis in large-scale computing environments. In: Supercomputing Conference (2006)
25. Agarwal, M., Anerousis, N., Gupta, M., Mann, V., Mummert, L., Sachindran, N.: Problem determination in enterprise middleware systems using change point correlation of time series data. In: NOMS (April 2006)
26. Chen, M.Y., Kiciman, E., Fratkin, E., Fox, A., Brewer, E.A.: Pinpoint: Problem determination in large, dynamic internet services. In: DSN, pp. 595–604 (2002)
27. Cohen, I., Goldszmidt, M., Kelly, T., Symons, J., Chase, J.: Correlating instrumentation data to system states: A building block for automated diagnosis and control. In: OSDI, pp. 231–244 (December 2004)

Maintenance of Monitoring Systems Throughout Self-healing Mechanisms

Clarissa Cassales Marquezan[1,2], André Panisson[1], Lisandro Zambenedetti Granville[1], Giorgio Nunzi[2], and Marcus Brunner[2]

[1] Federal University of Rio Grande do Sul - Porto Alegre, Brazil
{clarissa, panisson, granville}@inf.ufrgs.br
[2] NEC Europe Network Laboratories - Heidelberg, Germany
{marquezan, nunzi, brunner}@nw.neclab.eu

Abstract. Monitoring is essential in modern network management. However, current monitoring systems are unable to recover their internal faulty entities forcing the network administrator to manually fix the occasionally broken monitoring solution. In this paper we address this issue by introducing a self-healing monitoring solution. This solution is described considering a scenario of a monitoring system for a Network Access Control (NAC) installation. The proposed solution combines the availability provided by P2P-based overlays with self-healing abilities. This paper also describes a set of experimental evaluations whose results present the tradeoff between the time required to recover the monitoring infrastructure when failures occur, and the associated bandwidth consumed in this process. Based on the experiments we show that it is possible to improve availability and robustness with minimum human intervention.

1 Introduction

Network and service monitoring is an activity essential to identif problems in underlying IT communication infrastructures of modern organizations. Monitoring is typically materialized by systems that periodically contact elements (*e.g.*, network devices and services) to check their availability and internal status. A monitoring system may be simple (like the Multi Router Traffic Grapher (MRTG) [1]) or complex, being composed of as diverse entities as monitors, agents, and event notifiers. The information collected and processed by monitoring systems enables human administrators, responsible for managing the IT infrastructure, to identify (and possibly predict) problems, and thus react in order to keep the managed infrastructure operating in a proper way.

Monitoring systems must run uninterruptedly to ensure that failures in the managed elements are detected. Problems in the monitoring systems break the monitoring process and can lead the human administrator to believe that the managed elements are working properly even when they are not. Robust monitoring systems should thus employ mechanisms not only to identify failures on the managed infrastructure, but also to recover the faulty monitoring solution itself. Currently, however, most monitoring systems force the administrator to manually recover the occasionally broken solution. Such a manual approach may not drastically affect the monitoring of small networks, but in larger infrastructures the approach will not scale and should be replaced efficient alternatives. The self-managed approach is one alternative emerging as a solution for the manual approach. Typically, a self-managed system is built on top of self-* features capable to reduce the human intervention and provide more efficient results.

In this paper we address the problem of monitoring systems that lack self-healingness feature by considering the example of a Network Access Control (NAC) [2] installation. A NAC secured network is composed of devices and services (*e.g.*, routers, firewalls, RADIUS servers) that control how users and devices join the network. Traditional monitoring systems (*i.e.*, without self-healing support) fail to protect NAC, for example, in two situations. First, consider a crashed RADIUS server whose associated RADIUS monitor crashed too. In this case, the administrator reacts to the RADIUS problem only when users complain about unsuccessful login attempts. Worse than that, however, is the second situation. Suppose a failure in the *rogue user* service responsible for detecting unregistered devices, and another failure in the monitor associated to it. In this case, unregistered devices will silently join the network without generating user complains. In contrast to the first situation, the "silent failure" remains because no signal is issued either by network users or, and most seriously, by the now broken monitoring system.

Recent researches on autonomic management certainly present self-* concepts that could be used to address the aforementioned problems. Such researches, however, take a mostly abstract approach and rarely touch concrete implementation issues. In this paper, in turn, we investigate the employment of self-* features to actually implement, deploy, and evaluate a self-healing monitoring system able to recover from internal failures without requiring, at some extend, human intervention. The goal of our research is to understand the advantages and drawbacks of using self-* features in a real scenario of a service monitoring system.

The monitoring elements of our solution implement two main processes: regular monitoring (to monitor final devices and services) and recovery (to heal the monitoring system). We evaluate our solution in terms of recovery time when fail-stop crashes occur in the monitoring system. In addition, we also measure the traffic generated by the communication between the elements of our solution. This leads us to the contribution of showing the tradeoff between the recovery time and associated network traffic. The determination of such tradeoff is important because it shows when a faster recovery process consumes too much network bandwidth. On the other side, it also shows when excessively saved bandwidth leads to services that remain unavailable longer.

The remainder of the paper is organized as follows. In Section 2 we review network monitoring systems in terms of self-* support, distribution, and availability. In Section 3 we introduce our self-healing architecture for NAC monitoring, while in section 4 we evaluate our proposal in an actual testing environment, presenting associated results and their analyses. Finally, the paper is concluded in Section 5.

2 Related Work

Although network and service monitoring is widely addressed by current investigations [3] [4] and products on the market [5], we review in this section solutions that are mainly related to the aspects of self-monitoring and self-healing on distributed monitoring.

Distributed monitoring are specially required in large-scale networks, like country or continental-wide backbones [6] [7]. Most of the research in this area propose complex monitoring system that generally identify internal failures and employ algorithms to reorganize itself without the failed components. In this sense, the solutions present a certain level of self-awareness and adaption, although self-healing is in fact not present, *i.e.*, failing entities are not recovered or replaced. It means that in scenarios where most of the monitoring entities crash, the monitoring systems stop working because no mechanism is employed to maintain the execution of the monitoring entities.

Yalagandula *et al.* [8] propose an architecture for monitoring large networks based on sensors, sensing information backplane, and scalable inference engine. The communication among the entities relies on a P2P management overlay using Distributed Hash Tables (DHTs). Prieto and Stadler [9] introduce a monitoring protocol that uses spanning trees to rebuild the P2P overlay used for the communications among the nodes of the monitoring system. Both Yalagandula *et al.* and Prieto and Stadler work can reorganize the monitoring infrastructure if failures are detected in monitoring nodes. After such reorganization the failing nodes are excluded from the core of the rebuilt monitoring infrastructure. Although reorganized, with a few number of nodes, the monitoring capacity of the system is reduced as a whole. Again, adaptation in the form of infrastructure reorganization is present, but proper self-healing it is not.

Few investigations in fact explicitly employ autonomic computing concepts in system monitoring. Chaparadza *et al.* [10], for example, combine self-* aspects and monitoring techniques to build a traffic self-monitoring system. The authors define that self-monitoring networks are those that autonomously decide which information should be monitored, as well as the moment and local where the monitoring task should take place. Nevertheless, such work does not define how the monitoring system should react in case of failures in its components. The meaning of self-monitoring in this case is different than the one of our work. While self-monitoring in Chaparadza's work means autonomous decision about the monitoring process, in our view self-monitoring is about detecting problems, through monitoring techniques, in the monitoring system itself.

Yangfan Zhou and Michael Lyu [11] present a sensor network monitoring system closer to our view of self-monitoring. The authors' contribution resides on the use of sensors themselves to monitor one another in addition to performing their original task of sensing their surrounding environment. Although self-monitoring is achieved, given the restrictions of the sensor nodes (*e.g.* limited lifetime due to low-capacity batteries) the system cannot heal itself by reactivating dead nodes.

Considering the current state of the art, there is a necessity for new approaches for self-monitoring systems. New proposals should explicitly include, in addition to self-awareness already available in the current investigations, self-healing support on the monitoring entities in order to autonomously keep the monitoring service up. In the next section we thus present our self-healing approach for monitoring infrastructures.

3 Self-healing Architecture for Monitoring Infrastructures

The self-healing architecture built in our investigation forms a P2P management overlay on top of the monitored devices and services. The usage of P2P functionalities in our architecture provides a transparent mechanism to enable communications target to publish, discover, and access management tasks inside the overlay. In this way, the control of such basic communications is delegated to the P2P framework used to implement our architecture. Furthermore, in a previous work we have presented that network management based on P2P can aggregate benefits, like reliability and scalability, on the execution of management tasks [12]. Now, we use P2P overlays to have the transparency on basic overlay operations, to distribute the identification of failures and also to provide scalability on the recovery process.

The overlay proposed in previous work is called ManP2P and its architecture has been described in the work of Panisson *et al.* [13]. In this current paper, we extend the ManP2P functionalities in order to explicitly support self-healing processes. We believe that combined, self-healing and P2P overlays can bring together a self-monitoring infrastructure able to address current problems on monitoring systems. In this section we review the ManP2P architecture, present self-healing extensions, and exemplifies their employment in the concrete scenario of a NAC installation.

3.1 P2P Management Overlay and Services

The collection of management peers forms the ManP2P management overlay. Each peer runs basic functions (*e.g.*, granting access to other peers to join the overlay or detecting peers that left the P2P network) to maintain the overlay structure. In addition, each peer hosts a set o *management services* instances that execute management task over the managed network. In our specific case, such tasks are monitoring remote equipments. A management service is available if at least one single instance of it is running on the overlay. More instances of the same service, however, must be instantiated in order to implement fault tolerance. Figure 1 exemplifies a scenario where management services (LDAP monitors, Web servers monitors, rogue user monitors) for a NAC installation are deployed on the ManP2P management overlay.

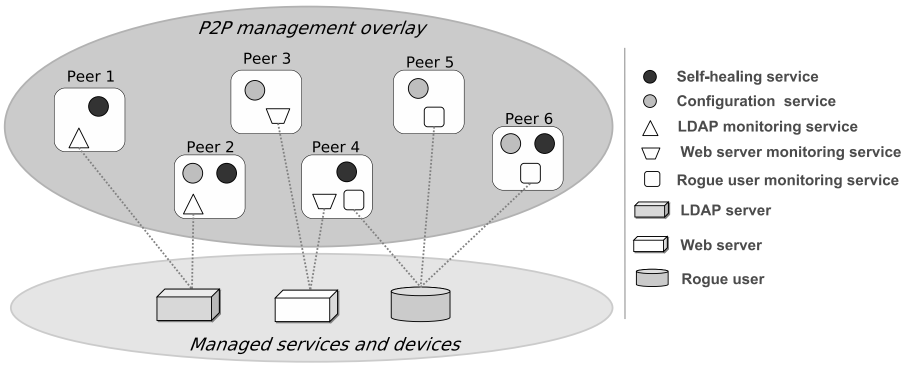


Fig. 1. NAC meta-monitoring infrastructure

In Figure 1, peers #1 and #2 host service instances, pictured as a triangle, that monitor an LDAP server. Peer #4, on its turn, contacts both the Web server and the rogue user service because it hosts management services to monitor these elements. The special services of self-healing and configuration, depicted as black and gray circles respectively, will be explained farther in this paper. Each peer, in summary, may host different services at one. In the extreme cases, there could exist peers with no management services (thus useless peers) or peers hosting one instance of each available management service (this possibly becoming an overloaded peer).

We consider that the a management service is able to heal itself if, after the crashing of some of its instances (possibly due to peers crash), new instances become available, thus recovering the service and guaranteeing its availability. In order to cope with that, two functions must be supported: failure detection and service instance activation.

3.2 Failure Detection

Failures in a management service are detected by a self-monitoring procedure where each service instance, in intervals of t seconds, sends a signal (heartbeat) to all other instances of the same service to inform that the former is running. Self-monitoring, in this sense, means that there is no external entity monitoring the instances of a management service deployed inside the overlay. Indeed, the instances of the management service themselves can monitor their liveness throughout the heartbeat messages. So, if one instance crashes, the other instances will miss the former's heartbeats and then will initiate the process to recover this instance, as it will be explained later on this paper.

Heartbeats that get lost in the network may wrongly suggest the unavailability of a service instance. Instead of immediately assuming an instance as down given the lack of a heartbeat, it first becomes suspect by the other instances. In order to double check the availability of the suspicious instance, one of the other alive instance tries to contact the suspicious instance back. If no contact is possible, the suspicious instance is finally declared unavailable. Assuming s as the time spent to double check the availability of a suspicious instance, the maximum detection time is $td = t + s$.

The distribution of heartbeats from one service instance to all others is accomplished using group communications. At the network level, in the best case, group communication is supported by multicast communications. In this case, the number of heartbeat messages h issued by i service instances in t seconds will be $h = i$. However, if multicasting is not available, the notifying service instance is forced to send, via unicast, copies of the same heartbeat to all other instances. In this case, the number of messages will be $h = i^2 - i$. In this way, the presence of multicasting directly influences the network traffic generated by the failure detection function.

Failure detection is essentially a consensus problem. Solutions on this topic, coming from the dependability field, could be employed and formalisms used to model and validade our detection approach. Instead of that, however, we preferred to use the practical approach of actually implementing the aforementioned function. Although no formal proof is provided, our experiments have shown that this approach is effective in detecting failures in the management service instances.

Table 1. Service policy repository

Management service	*Minimum instances*	*Activate instances*
LDAP monitor	2	1
Web server monitor	2	2
Rogue user monitor	2	1

3.3 Service Instance Activation and Policies

Instance activation is crucial to recover the management service that just lost some of its instances. It is on instance activation that the self-healing and configuration services, presented in Figure 1, play a key role.

Once an instance detects a remote crashed one, it notifies the *self-healing service* that determines how many, if any, new instances of the faulty service must be activated. To do so, the self-healing service internally checks a repository of service policies that describes, for each management service, the minimum number of instances that must be running, as well as the number of new instances that must be activated once the minimum boundary is crossed.

Table 1 shows the service policy repository for the NAC installation of Figure 1. As can be observed, the LDAP monitoring service must have at least 2 instances running. In cause of failure, another new one instance must be activated. In the case of the Web server monitor, on the other hand, although 2 instances are running, whenever activation is required 2 other new instances will be initiated. If the number of remaining running instances of a services is still above the minimum boundary, the self-healing service ignores the faulty service notifications. For example, in the case of the rogue user monitor from Figure 1, if a single instance crashes no action will be executed because the remaining 2 instances do not cross the minimum boundary. Although it is outside the scope of this paper stressing the administration and usage of management service policies (refer to the work of Marquezan *et al.* [14] for that), we assume that policies are defined by the system administrator and transferred to the self-healing service instances long before any failure occurred in the P2P management overlay.

Once required, the self-healing service tries to activate the number of new instances defined in the service policy by contacting the *configuration service*. Such configuration service is then responsible for creating new instances of the faulty service on peers that do not have those instances yet. A peer hosting solely a configuration service can be seen as an spare peer ready to active new instances of any service in failure.

Different than the failure detection function, instance activation is performed outside the group of instances that implement the failing management service. That is so because decoupling the instance activation function from the services that require them allow us to more flexibly deal with the number of components for each function. That directly impact on the number of message exchanged in the overlay.

So far, we have defined a self-healing architecture that extends the ManP2P functionalities. However, to ensure that the failure detection and instance activation functions work properly, two requirements must be filled on the P2P management overlay. First, each management service (including the self-healing and configuration services) must run at least 2 instances in order to detect and recover problems on the management service. That is so because a single faulty instance cannot react itself if it is crashed, then at least another instance is required. Second, each peer must not host more than one instance of the same management service in order to avoid several instances of that service crashing if the hosting peer crashes too. We assure that the maintenance of the monitoring infrastructure can be accomplished while these requirements are fulfilled.

3.4 System Implementation

As mentioned before, our architecture extends the ManP2P system. The implementation of our architecture in an actual monitoring system is than based on the previous code of ManP2P. Figure 2 depicts the internal componentes of a peer of our solution.

Components are divided by the *core peer plane* and *management service plane*. The core peer plane's components are responsible for controlling the communication mechanisms between peers. At the bottom the *JXTA* and *network multicast* components implement group communication using unicast (via JXTA) or network multicast. On top of them, the *group manager* and *token manager* components control, respectively, group membership and load balancing (via a virtual token ring). Messages are handled by the *message handler* component that interfaces with Axis2 to communicate with the management service plane's components. A ManP2P component on the top of the core peer plane is used to implement complementary functionalities that are not inside the scope of this paper.

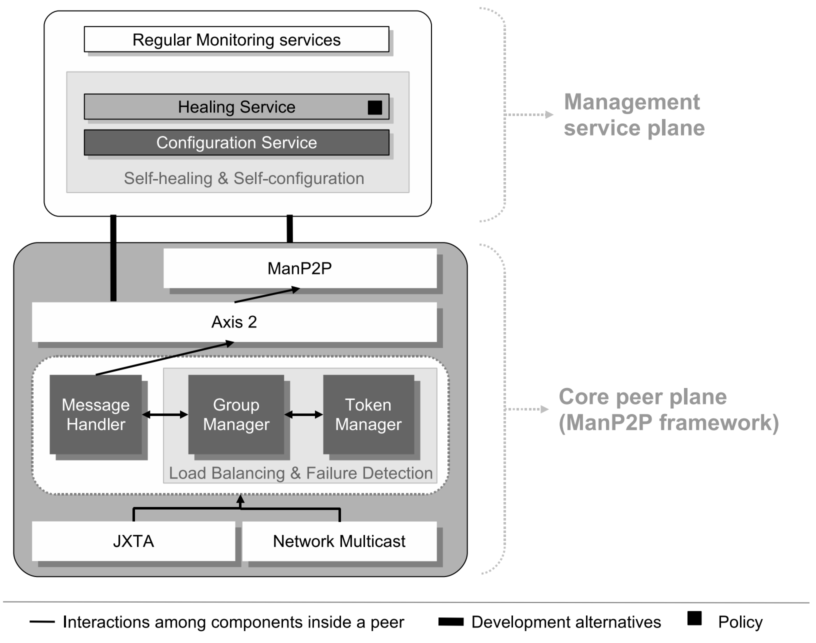

Regular Monitoring services
Healing Service
Configuration Service
Self-healing & Self-configuration
Management service plane
ManP2P
Axis 2
Message Handler
Group Manager
Token Manager
Load Balancing & Failure Detection
JXTA
Network Multicast
Core peer plane (ManP2P framework)
Interactions among components inside a peer
Development alternatives
Policy

Fig. 2. Meta-monitoring architecture

At the management service plane the regular monitoring services are found. Although located in this plane, monitoring services themselves do not monitor remote instances for fault detection; this verification is in fact performed by the group manager component. That is so because we wanted the self-monitoring function to be native in any peer, freeing the developer of new management services to concentrate their efforts on the management functionalities he/she is coding without worrying about self-monitoring. At the management service plane the self-healing and configuration services are also found. As mentioned before, they are responsible for activating new instances of monitoring services when required. The black little square inside the self-healing service represents the policies that define the minimum number of instances of each management service, as well as the number of new instances that must be activated. Peers and internal monitoring services have been coded in Java using Axis2, JXTA, and ManP2P previously developed libraries. Monitoring services have been specifically developed as dynamic libraries that can be instantiated when required be a hosting peer.

4 Experimental Evaluation

In our experimental evaluation we measured the recovery time and the generated network traffic when fail-stop crashes occur in peers of the proposed self-healing monitoring infrastructure. We evaluate the effects of such failures considering variations on: (a) the number of simultaneously crashing peers, (b) the number of peers in the management overlay, and (c) the number of management services running on the overlay.

We have run our experiments in a high performance cluster, called LabTec from the GPPD research group at UFRGS [15], from which we used 16 nodes to host the management peers of our architecture. The recovery time and the generated traffic have been measured capturing the P2P traffic and timestamping it using a packet capture `tcpdump` software. Traffic volume is calculated considering the headers and payload of all packets generated by the system operations. Recovery time has been measured 30 times for each experimental case and computed with a confidence interval of 95%.

Although the size of P2P systems is typically of scales much higher than 16 nodes, we emphasize here that we do not believe that, in an actual management scenario of a single corporation, administrators would use a large number of managing nodes. We thus assume that 16 peers are sufficient for most actual management environments. Over the P2P management overlay we deployed up to 12 different NAC management services (namely, monitors for LDAP, DNS, DHCP, Radius, data base, Web servers, rogue user, firewall, proxy, access point, switches, and routers), in addition to the self-healing and configuration special services required in the recovery process. The single service policy enforced in all management services of our experiments defines that at least 2 instances per service must be running and, in case of failures, just 1 another instance must be activated per crashed instance.

Considering the above, two main sets of experiments have been carried out: multiple crashing peers, and variable number of peers and services. These experiments and associated results are presented in the next subsections.

4.1 Multiple Crashing Peers

The first experiment was designed to check the performance of the self-healing monitoring architecture when the number of simultaneously crashing peers hosting management services increases until the limit where half of them are broken. In addition, we want to check whether the number of instances of the self-healing and configuration services influences the recovery time and generated traffic.

For this set of experiments, we used to following setup: 12 management services are always deployed, each one with 2 instances running on the overlay. The total 24 service instances (*i.e.*, 12 × 2) are placed along 8 peers, each one thus hosting 3 (*i.e.*, 24 ÷ 8) service instances. The number of crashing peers varies from 1 to 4. Since each peer hosts 3 instances, the number of crashing instances varies from 3 (12.5%) to 12 (50%), out of the total of 24 instances. Additional 4 peers have been used to host the self-healing and configuration services. Their varying number of instances has been organized, in pairs of self-healing/configuration, as follows: 2 and 4 instances, and 4 and 4 instances. Finally, we consider that group communication support is implemented interchangeably using multicast and unicast.

Figure 3 shows in seconds the time taken by the monitoring system to detect and activate new instances of the crashing services using the "spare" cluster nodes that host the configuration service. The first occurrence of 3 crashing services correspond to the situation where 1 peer fails; 6 crashing services correspond to 2 failing peers, and so on. No value is provide in 0 (zero) because with no failing peers there will not be any crashing service. Figure 4, in its turn, presents the network traffic generated by the management overlay in this recovery process. In this case, for 0 (zero) there exists an associated network traffic because, in the self-monitoring process, heartbeat messages are constantly sent regardless the presence or not of a failure.

The recovery time as a function of the number of crashing peers stayed mostly constant. With that we can conclude that the system scales well considering a management scenario of 16 nodes. There is a little variance on the recovery time as a function of the self-healing and configuration services. In fact, such difference is the result of

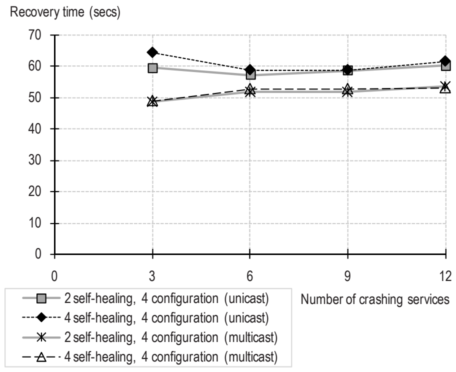


Fig. 3. Recovery time with multiple crashing

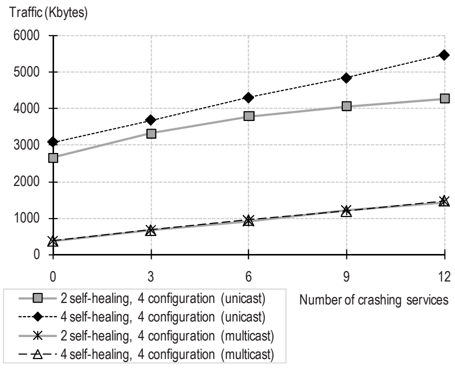


Fig. 4. Traffic to recover crashing peers

employing multicast or unicast. When peers use multicasting they quickly become aware of changes in the system, and can rather react faster. Using unicast, however, more messages are sent, delaying the communication and, as a consequence, the reactions. In summary, the recovery time is not strongly influenced either by the self-healing and configuration services or by the number of crashing services. There is, however, a little influence from the use of multicast or unicast in the group communication support.

Network traffic, in its turn, presents a stronger influence of multicast or unicast support. As can be observed in Figure 4, multicast-based communications saves more bandwidth, which is expected. The important point to be observed, however, is that with the increasing number of crashed services the traffic generated to recover them is closely linear, but with doubling the number of failures, the traffic generate does not double together. Although not so efficient as in the case of recovery time, the bandwidth consumption is still scalable in this case. Putting these two parameters together and observing the graphs, if multicasting is used the number of self-healing and configuration services and the number of crashing peers do not influence the recovery time, and slightly increase the bandwidth consumption. In the case of unicast, however, the option of employing 2 self-healing instances instead of 4 is better, because this setup reacts slightly faster yet generating less traffic.

4.2 Varying Number of Peers and Services

The second experiment shows the relationship between recovery time and generated traffic when single crashes occur (which tends to be more frequent than multiple crashes) but the number of peers and services varies. We consider the recovery process when the number of management services increases (from 1 to 12, *i.e.* from 2 to 24 instances) over three setups where 2, 6, and 12 peers are used to host the management services. In addition to single crashes, we also fixed the number of 2 self-healing and 2 configuration services instances, hosted by 2 peers. We did so because, as observed before, the number of such instances few impacts on the recovery time.

In Figure 5, where the recovery delay is presented, services communicating via multicast are depicted with dashed lines, while services using unicast are depicted with solid

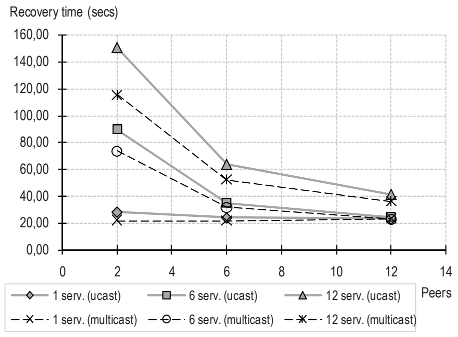


Fig. 5. Recovery time for multiple peers

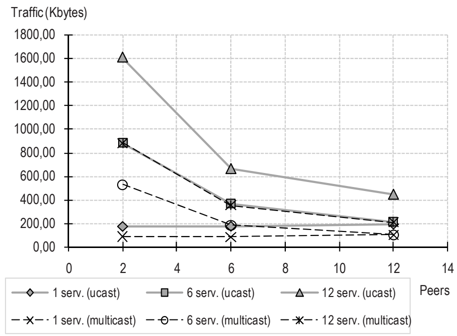


Fig. 6. Recovery traffic with multiple peers

gray lines. The recovery time when only 2 peers are employed is usually higher because each of the 2 peers hosts more service instances. When one of the peers crashes, more instances need to be activated. On the other extreme, with 12 peers, each peer hosts less services, leading to the situation where a crashing peer actually triggers the activation of less service instances.

The fact that more instances need to be activated as the result of a more load peer can be observed in Figure 6, that shows the traffic generated to recover the system. Again, multicast communications save more bandwidth than unicast, as expected. However, it is important to notice now that the number of services in each peer influences too. For example, 6 instances running on the same peer (line “6 serv. multicast”, with 2 peers in the x axis) despite being multicast still takes longer and generates more traffic to recover the system than the case where, via unicast, only 1 services is deployed (line “1 serv. unicast”, with 2 peers in the x axis).

This confirms that the number of peers and service instances must be similar in order to recover more promptly the system without generating too much traffic. If an administrator is restricted in terms of peers available, he/she must try to restrict the number of services employed as well. If new services are required, however, the option of also increasing the number of peers should be considered.

Now considering the whole picture, administrators should not worry about simultaneous crashes nor the number of self-healing and configuration services. Increased multiple crashes are more scare, and even if they happen the system is able to recover reasonably fast. As observed, the number of serf-healing and configuration services does not affect the overall performance of the system. However, administrator should do pay attention to the number of available peers and service instances, as mentioned before. Finally, the employment of multicast and unicast in the group communication mechanism influences in the recovery time (less) and the generated traffic (more). Choosing multicast whenever possible helps to improve the response time of the system. Unfortunately multicasting is not always available, which forces the administrator to use unicast to implement group communication.

5 Conclusions and Future Work

We have presented in this paper the design and evaluation of a P2P-based self-healing monitoring system employed in a NAC environment. The solution achieves self-healing capacity by splitting in two different processes the functions of failure detection and system recovery. Failure detection is executed inside management services that monitor final devices, while system recovery relies on special services called self-healing (that decides when new service instances must be activated) and configuration (that activates the new service instance as an reaction for the self-healing service decision).

The results of our experimental evaluations allow us to conclude that the number of instances of the self-healing and configuration service is not a major player in the performance of the system. They also permit us to state that simultaneously crashes on the management services does not influences so expressively the system performance either. A network administrator willing to employ a self-healing monitoring solution should not concentrate his/her efforts in finding an ideal number of self-heling and configuration services. Our experiments employed 2 and 4 instances, respectively, and the system response was satisfactory. The fact that must be observed, however, is the group communication solution available on the managed network: multicast turns recovery faster while consuming less network bandwidth. Unfortunately, IP multicasting can not always be provided, and unicast ends up being chosen in such situations.

The most important aspects that must be observed in a self-healing monitoring solution is the number of peers employed in the P2P management overlay and the number of service instances deployed. With few instances, there is no need for several peers. On the other hand, with a large number of instances the number of peers should grow consistently, otherwise, on the occurrence of a failure, the recovery time will be higher and more network bandwidth is consumed by the intensive P2P traffic generated.

Currently, we are working on the optimization of the detection mechanism because the current version of it is responsible for a considerable amount of generated traffic. Another future work is the investigation about how service policies impact on the consumed network bandwidth and recovery time of our system.

Acknowledgement

Thanks to Dominique Dudkowski and Chiara Mingardi for contributing with this paper, and also to the Network and Support Division team at Data Processing Center of UFRGS for the experience exchanged on the development of the UFRGS NAC system.

This work was partly funded by the Brazilian Ministry of Education (MEC/CAPES, PDEE Program, process 4436075), and by the European Union through the 4WARD project in the 7th Framework Programme. The views expressed in this paper are solely those of the authors and do not necessarily represent the views of their employers, the 4WARD project, or the Commission of the European Union.

References

1. Oetiker, T.: MRTG - The Multi Router Traffic Grapher. In: LISA 1998: Proceedings of the 12th USENIX conference on System administration, Berkeley, CA, USA, USENIX Association, pp. 141–148 (1998)
2. López, G., Cánovas, O., Gómez, A.F., Jiménez, J.D., Marín, R.: A network access control approach based on the AAA architecture and authorization attributes. J. Netw. Comput. Appl. 30(3), 900–919 (2007)
3. Perazolo, M.: A Self-Management Method for Cross-Analysis of Network and Application Problems. In: 2nd IEEE Workshop on Autonomic Communications and Network Management (ACNM 2008) (2008)
4. Trimintzios, P., Polychronakis, M., Papadogiannakis, A., Foukarakis, M., Markatos, E., Oslebo, A.: DiMAPI: An Application Programming Interface for Distributed Network Monitoring. In: Proceedings. 10th IEEE/IFIP Network Operations and Management Symposium, 2006. NOMS 2006, pp. 382–393 (2006)
5. Packard, H.: Management Software: HP OpenView (2008), `http://www.openview.hp.com/`
6. Agarwal, M.K.: Eigen Space Based Method for Detecting Faulty Nodes in Large Scale Enterprise Systems. In: IEEE/IFIP Network Operations and Management Symposium (NOMS 2008) (2008); CDROM

7. Varga, P., Moldován, I.: Integration of Service-Level Monitoring with Fault Management for End-to-End Multi-Provider Ethernet Services. IEEE Transactions on Network and Service Management 4(1), 28–38 (2007)
8. Yalagandula, P., Sharma, P., Banerjee, S., Basu, S., Lee, S.J.: S3: a scalable sensing service for monitoring large networked systems. In: INM 2006: Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management, pp. 71–76. ACM Press, New York (2006)
9. Prieto, A.G., Stadler, R.: A-GAP: An Adaptive Protocol for Continuous Network Monitoring with Accuracy Objectives. IEEE Transactions on Network and Service Management 4(1), 2–12 (2007)
10. Chaparadza, R., Coskun, H., Schieferdecker, I.: Addressing some challenges in autonomic monitoring in self-managing networks. In: 13th IEEE International Conference on Networks, p. 6 (2005); CDROM
11. Zhou, Y., Lyu, M.R.: An Energy-Efficient Mechanism for Self-Monitoring Sensor Web. In: 2007 IEEE Aerospace Conference, pp. 1–8 (2007)
12. Granville, L.Z., da Rosa, D.M., Panisson, A., Melchiors, C., Almeida, M.J.B., Tarouco, L.M.R.: Managing Computer Networks Using Peer-to-Peer Technologies. IEEE Communications Magazine 43(10), 62–68 (2005)
13. Panisson, A., Melchiors, C., Granville, L.Z., Almeida, M.J.B., Tarouco, L.M.R.: Designing the Architecture of P2P-Based network Management Systems. In: Proceedings. IEEE Symposium on Computers and Communications (ISCC 2006), pp. 69–75. IEEE Computer Society, Los Alamitos (2006)
14. Marquezan, C.C., dos Santos, C.R.P., Nobre, J.C., Almeida, M.J.B., Tarouco, L.M.R., Granville, L.Z.: Self-managed Services over a P2P-based Network Management Overlay. In: Proc. 2nd Latin American Autonomic Computing Symposium (LAACS 2007) (2007)
15. GPPD: Parallel and Distributed Processing Group – GPPD (2008), `http://gppd.inf.ufrgs.br/new/`

Author Index