KAIST Research Series

Chan-Hyun Youn
Min Chen
Patrizio Dazzi

Cloud Broker and Cloudlet for Workflow Scheduling

KAIST

Springer

KAIST Research Series

More information about this series at http://www.springer.com/series/11753

Chan-Hyun Youn · Min Chen
Patrizio Dazzi

Cloud Broker and Cloudlet for Workflow Scheduling

Springer

Chan-Hyun Youn
School of Electrical Engineering
KAIST
Daejeon
Korea (Republic of)

Patrizio Dazzi
Area della Ricerca di Pisa
ISTI-CNR
Pisa
Italy

Min Chen
Embedded and Pervasive Computing (EPIC) Lab
Huazhong University of Science and Technology
Wuhan, Hubei
China

ISSN 2214-2541 ISSN 2214-255X (electronic)
KAIST Research Series
ISBN 978-981-10-5070-1 ISBN 978-981-10-5071-8 (eBook)
DOI 10.1007/978-981-10-5071-8

Library of Congress Control Number: 2017943825

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

Chapter 1
Integrated Cloud Broker System and Its Experimental Evaluation

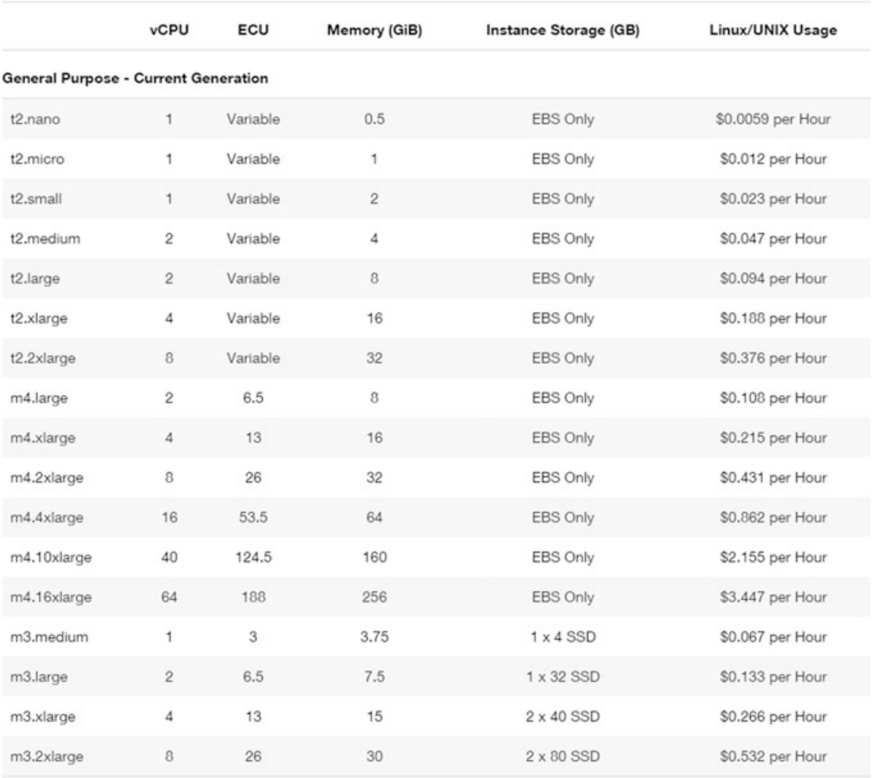
1.1 Cloud Broker System Overview

In distributed computing environment, there are a large number of similar or equivalent resources provided by different service providers. These resources may provide the same functionality, but optimize different Quality of Service (QoS) metrics. These computing resources are managed and sold by many different service providers [1]. Service providers offer necessary information about their services such as the service capability, and the utility measuring methods and charging policies, which will be later referred to as the "resource policy" in this book. Each resource policy bears a tuple of two components, such as (capability, price). For capability, we model the resource capability as a set of QoS metrics which include the CPU type, the memory size, and the storage/hard disk size. Figure 1.1 is an example of the various resources types and their charging policies provided by Amazon EC2. Pricing is per instance-hour consumed for each instance, from the time an instance is launched until it is terminated [2].

Users can outsource their local applications or directly access some web-service to leverage the remote computing resources in the cloud and buy their preferred services from the service provider. However, usually there are many different service types to choose from and the user probably will not make an optimized decision based on their limited knowledge. Moreover, this may cause an inefficient execution of their application with longer elapsed time and more monetary cost than that it actually needs. In addition, different users or applications may have different expectations and requirements.

It is necessary to develop a broker as an intermediate negotiator between the user and the service provider, which orchestrates the application policy on the user's side and the resource policy on the service provider's side. The broker functions to bridge the user with the service provider while hiding many workflow mapping details so that the user can utilize the vast resource pool within the internet in a transparent and easy way. Moreover, the broker can offer different service levels

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_1



	vCPU	ECU	Memory (GiB)	Instance Storage (GB)	Linux/UNIX Usage
General Purpose - Current Generation					
t2.nano	1	Variable	0.5	EBS Only	$0.0059 per Hour
t2.micro	1	Variable	1	EBS Only	$0.012 per Hour
t2.small	1	Variable	2	EBS Only	$0.023 per Hour
t2.medium	2	Variable	4	EBS Only	$0.047 per Hour
t2.large	2	Variable	8	EBS Only	$0.094 per Hour
t2.xlarge	4	Variable	16	EBS Only	$0.188 per Hour
t2.2xlarge	8	Variable	32	EBS Only	$0.376 per Hour
m4.large	2	6.5	8	EBS Only	$0.108 per Hour
m4.xlarge	4	13	16	EBS Only	$0.215 per Hour
m4.2xlarge	8	26	32	EBS Only	$0.431 per Hour
m4.4xlarge	16	53.5	64	EBS Only	$0.862 per Hour
m4.10xlarge	40	124.5	160	EBS Only	$2.155 per Hour
m4.16xlarge	64	188	256	EBS Only	$3.447 per Hour
m3.medium	1	3	3.75	1 x 4 SSD	$0.067 per Hour
m3.large	2	6.5	7.5	1 x 32 SSD	$0.133 per Hour
m3.xlarge	4	13	15	2 x 40 SSD	$0.266 per Hour
m3.2xlarge	8	26	30	2 x 80 SSD	$0.532 per Hour

Fig. 1.1 An example of the various resources types and their charging policies [2]

according to different users' requirements and provide corresponding QoS guarantees respectively. The broker has both the functionalities of task management and resource management. For each application execution, there are three tightly interactive roles–users, broker, and service provider [3]. An application is submitted to the cloud by users, scheduled by the task scheduling scheme based on the interaction between the broker and the resource providers, and finally executed by the providers. There is no particular interaction between the users and the service providers, which are decoupled by the broker as represented in Table 1.1.

1.1.1 Service Provider and User Perspectives

Service providers mean every entity which enables cloud processing procedures, and include the followings: Resource providers which provide IaaS such as Terremark, Savvis, GoGrid, Amazon, and Rackspace; PaaS and SaaS resource providers; Management service providers which provides backups, fault-tolerance,

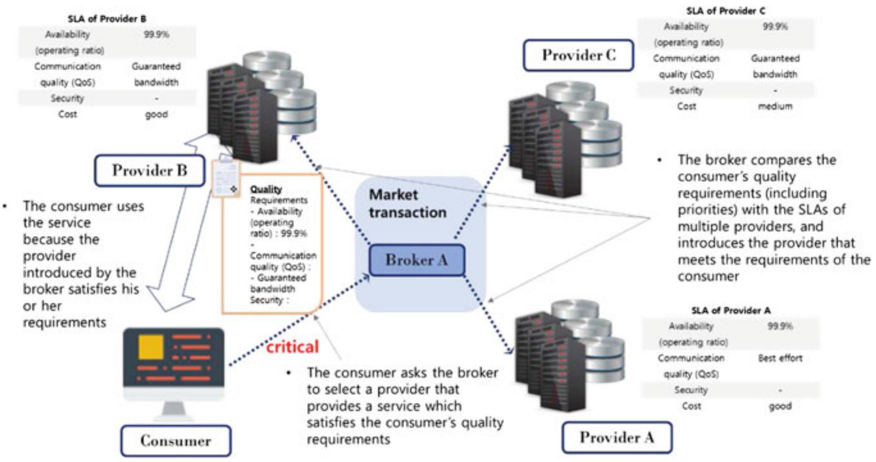
Table 1.1 Cloud resource management matters in diverse perspective

	Location	– Physical resources location
Cloud service provider	Service cost	– Computation cost – Storage cost – Software cost – Network cost
	Infrastructure cost	– Power supply – Cooling system – Electric cost
Cloud resource broker	Workflow engine	– Workflow scheduler – Policy adapter – Resource allocation service
	Resource manager	– VM allocation service – Resource monitoring – Leasing manager – Capacity optimizer
	Data scheduling	– Load balancer – Data provisioning service
Cloud service user	Load balancing	– VM assignment – Load balancing policy
	QoS	– Service availability – Performance – Security
	Application scaling policy	– Scale up/out – Service monitoring, SLA

and monitoring. The cost for providing resource can be classified as server cost (computation, storage, software), infrastructure cost (power supply and cooling), electric cost, network cost (link, transmission, equipment), and the cost for consuming services include everything which is served. Currently, many companies adopt cloud computing, and cloud computing is also attractive to universities, research institutes which needs resource for high performance computing.

1.1.2 Cloud Resource Broker Perspectives

Brokers connect service providers and consumers, and control cloud services to guarantee QoS, improve application service performance in terms of …, etc. Because there is no service provider which can satisfy every requirement of different consumers, the role of selecting proper service providers to serve consumers' requests is extremely important. In addition, general consumers may have difficulty in selecting the optimal resource for executing their applications, and brokers can help consumers to solve this problem. Figure 1.2 shows an example of functions of cloud brokers in inter-cloud environment. As shown in the figure, the broker is required to select the most appropriate providers by comparing providers' Service


SLA of Provider B
Availability (operating ratio) 99.9%
Communication quality (QoS) Guaranteed bandwidth
Security -
Cost good
Provider B
Quality Requirements
- Availability (operating ratio) : 99.9%
- Communication quality (QoS) :
- Guaranteed bandwidth
Security :
Market transaction
Broker A
critical
Consumer
Provider C
SLA of Provider C
Availability (operating ratio) 99.9%
Communication quality (QoS) Guaranteed bandwidth
Security -
Cost medium
SLA of Provider A
Availability (operating ratio) 99.9%
Communication quality (QoS) Best effort
Security -
Cost good
Provider A

- The consumer uses the service because the provider introduced by the broker satisfies his or her requirements
- The consumer asks the broker to select a provider that provides a service which satisfies the consumer's quality requirements
- The broker compares the consumer's quality requirements (including priorities) with the SLAs of multiple providers, and introduces the provider that meets the requirements of the consumer

Fig. 1.2 Functional example of cloud resource broker in inter-cloud

Level Agreement (SLA) and consumers' QoS for consumer's requests. For this purpose, the consumers provide the information of QoS in the figure, and the broker provides the provider list which satisfies the constraints. Therefore, the consumers can use the most appropriate resource.

1.1.3 Bipartite SLAs Between Stakeholders

The three roles, say user, broker, and cloud service provider, need to express their requirements and facilitate scheduling decision to further achieve their objectives [4]. We therefore utilize SLA that is usually defined in the community as a business agreement between each two of them to create the common under-standing about services, responsibilities, and others [5]. There will be two bipartite SLAs that are represented by the SLA type I relation (SLA1) which is established between users and the broker, as well as the SLA type II relation (SLA2) which is established between the broker and providers. Figure 1.3 represents SLA1 and SLA2 between users, broker, and service providers.

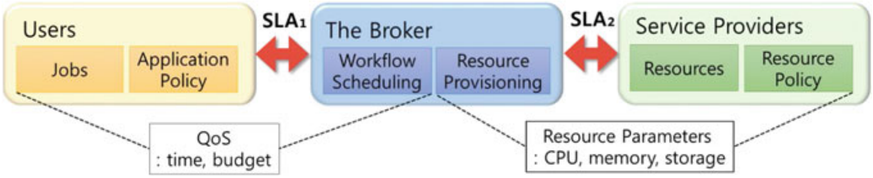


Fig. 1.3 Bipartite SLAs among users, the broker, and service providers

Users and jobs. When a user wants to use the computational resources to execute a workflow application, the user first submits the workflow with SLA1 to the broker. There are many potential QoS requirements specified in the SLA1 for the user. Current existing systems are mostly restricted to best-effort optimizations for time-based criteria, such as reducing overall execution time or maximizing bandwidth; while in our proposed system, we plan to impose the QoS guarantee mechanism. The QoS parameters declared in the SLA1 include the time constraint and the budget constraint for executing the whole workflow application. Specifically, for each sub-task within the workflow application, we identify the SLA1 between the user and the broker and the corresponding QoS parameters. This SLA1 is mapped by the broker to SLA2 and the SLA2 then will be used to make a resource provisioning schedule by the broker.

Broker and scheduling scheme. The broker is the middleware between the user and the resource provider by providing resource allocation service to satisfy users' requirements. In order to make scheduling decisions to satisfy SLA1, the broker interacts with resource providers to lease resources by using the SLA2 between the broker and the service providers which is defined as a 3-tuple with three resource parameters—CPU type, memory size, and storage/hard disk size.

Providers and resources. From the scheduling point of view, each resource can be modeled with two parameters: capability and job queue. The capability is the computational speed for reconfiguring resources, which has a unit of million instructions per second, and can be considered to be closely related to three parameters—CPU type, memory size, and storage/hard disk size. The job queue of a resource keeps an ordered set of jobs scheduled but not yet executed.

In this book, the technical part of the SLA is decoupled into two bipartite SLAs that are represented by the SLA type I relation (SLA1) which is established between users and the broker, as well as the SLA type II relation (SLA2) which is established between the broker and providers. The two bipartite SLAs' values together define the QoS offered to a user. The broker incorporates a QoS–enabled workflow management module and an adaptive resource provision module. The workflow management module will perform a mapping function from the SLA1 to a corresponding SLA2. from a resource provision module's perspective, QoS constraints declared in SLA1 from user's perspective will be mapped to resource parameters declared in SLA2. Resource provision module will then use the SLA2 to allocate suitable resources to execute the current sub-task in the workflow.

Therefore, users must be able to specify their requirements, either completion time or cost, for the whole workflow at design level. Then, the actions conducted by the broker must be chosen accordingly. Moreover, because different users or applications may have different expectations and requirements, the scheduling scheme must be dynamically chosen according to the QoS requirements. Therefore, a policy-based adaptive broker is proposed which has not only functions for integrated workflow management with resource management, but also multiple scheduling schemes for supporting different users' preferences.

Existing policy-based resource management systems [6, 7] manage the resources by considering the system and network status parameters like the CPU speed, memory size, storage capacity, I/O devices, band-width, delay and jitter. According to the dynamic change of resource status, the resource management system generates different optimized resource list based on the policy. However, from the user's point of view, other QoS factors like time, cost, fidelity, reliability, and security should be considered to be defined in the SLA. In other words, the existing work cannot directly satisfy user's most caring factors like time and cost. This requires an integrated management scheme to orchestrate the workflow management and the resource management. Another existing workflow management system [8] handles the SLA issues by allowing users to designate their wanted comparatively weight on cost and time. However, the system does not tackle the real SLA issue such as completion time or execution cost.

Therefore, the objective of the integrated model proposed in this book is to provide QoS guaranteed service for user submitted workflow by satisfying the user's SLA specification which includes direct and easy metrics such as time and/or budget constraint. The model should be aware of the dynamic change of resource status and adaptive to satisfy different requirements from users. We expect that our proposed policy-based adaptive broker can analyze the QoS parameters from SLA specification, and provision appropriate resources to sub-tasks based on such information.

1.2 VM Resource Management Schemes in Cloud Brokers

1.2.1 Resource Management System in Heterogeneous Cloud Environment

In current cloud computing ecosystem, there are many heterogeneous cloud environments. Therefore, the problems such as inter-compatibility and security can arise because of difficulty of interlock between various cloud platforms. Therefore, for appropriate use of content which is shared in huge organization by many users, methods to effectively manage resources in heterogeneous cloud datacenters are necessary and essential.

(1) **Resource management issue in heterogeneous cloud computing environment**

The resource management in cloud computing environment is the process to allocate appropriate computing, network, storage, and energy resource to the application. There are three roles in this environment: Cloud Service Provide, Cloud User, and End User. The cloud service provider operates a datacenter and provide the cloud resource in IaaS to the cloud user with a charge. The cloud user acts as a kind of broker to process the application to be executed by the end user with appropriate cloud resources. The end user requests the application execution to the cloud user. QoS which must be guaranteed at least are determined as a SLA between each other, as shown in Fig. 1.3. In this case, each role should apply the cloud resource management in order to guarantee the SLA and maximize its profit.

For example, the cloud service provider can guarantee the QoS such as Service availability, Performance, Security, and Data access as a SLA to the cloud user. On the other hand, the cloud user can guarantee the QoS such as Deadline, Fault tolerance as a SLA to the end user. The cloud service provider should minimize its cost such as the power consumption of a data-center and maximize its profit while guaranteeing the SLA required by end users. All schemes related to this object are called as cloud resource management scheme. The cloud resource management schemes are comprised of resource scheduling, virtual machine placement, resource profiling, resource usage prediction, and task management.

Because the available resource is limited in a single cloud system, cloud service providers are required to adopt inter-cloud. Therefore, resource management schemes between clouds should be developed. Practically, it is needed to provide convenient interface for interaction and guarantee QoS factors such as performance and availability, etc. (Fig. 1.4).

(2) **Heterogeneous resource service and interface**

Cloud computing enables high performance computing, massive data processing, and resource sharing by connecting network among various computing resource. Therefore, cloud computing can be a good solution to integrate distributed computing resource and provide resource/service collaboration. A cloud resource service interacts with other services with feedback loops. Figure 1.5 shows the example of cloud resource services. In Fig. 1.5a, cloud resource services are provided by the feedback loop (policing, alerting, negotiations, provision, transfer).

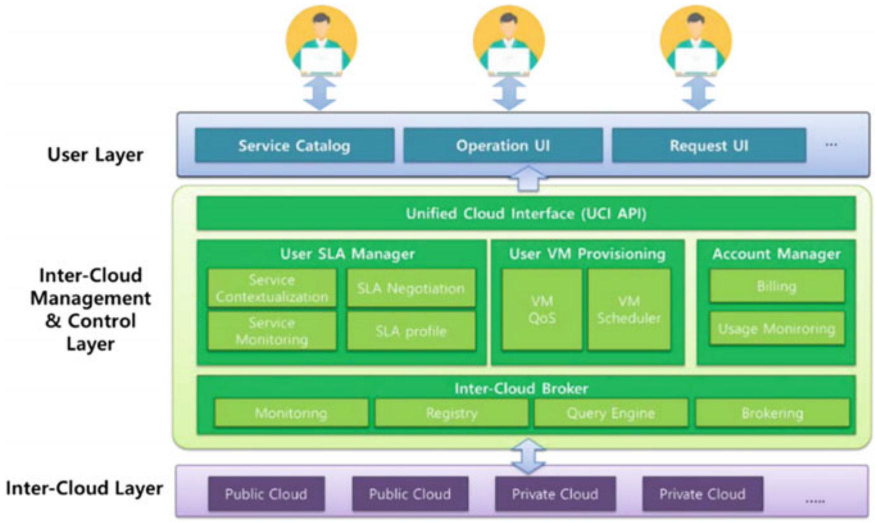


Fig. 1.4 Architecture of inter cloud resource Manager

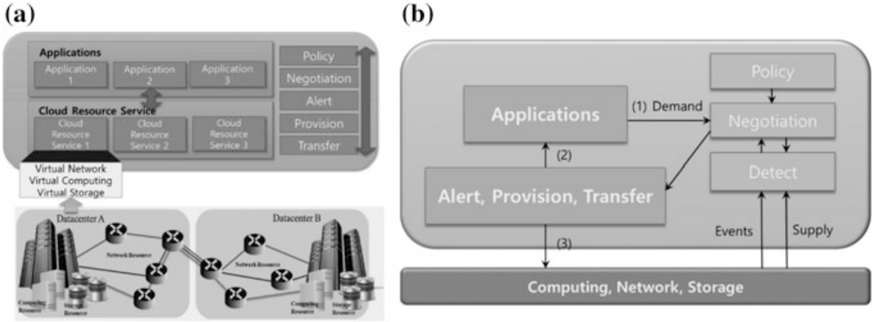


Fig. 1.5 Procedure of cloud resource provisioning service **a** enable block, **b** action among function blocks

Cloud brokers can activate the services, and the services are processed with service factors such as data rate, total data volume, data flow, etc.

1.2.2 Technical Requirement of Brokers for Heterogeneous Cloud Resource Management

Cloud providers should satisfy each consumer's service requirement to guarantee certain level of QoS. The QoS factors can be defined and differentiated by each kind of services (Table 1.2).

- Guarantee performance

Guaranteeing performance means consistently maintaining service performance in a certain level by distributing load to cloud systems to meet consumer's SLA even if several cloud systems are impossible to guarantee performance because of overload.

- Guarantee availability

Guaranteeing availability means consistently maintaining service provision even if there are faults in the cloud systems. To achieve it, it is necessary to recover service by migrating the services to cloud systems operated well. If recovery of every service is impossible, services may be recovered by the corresponding priorities. For example, cloud providers can recover services with high priority to guarantee required QoS of the services or change this part.

- Enhance convenience by service cooperation

Enhancing convenience by service cooperation means all services cooperating as if every related service is processed in the same place in the form of a one-stop service.

Table 1.2 SLA factors

SLA item		
Availability	Service availability, availability	Probability that service is available (scheduled service time—service delay)/ scheduled service time
	Average recovery time	Average time while recovering faults
	Service suspension time	Recovery time for expected faults
	Time of data recovery point	Time when data is recovered
Performance	Online response time	Response for online processing
	Online response time compliance ratio	Online transaction ration completed within deadline (%)
	Batch processing time	Response time for batch processing
	Batch processing time compliance ratio	Batch processing ration completed within deadline (%)
	Maximum number of processing tasks per unit time	Maximum number of processing tasks per unit time
	Compliance ratio of maximum processing tasks per unit time	Compliance ratio of maximum processing tasks per unit time (%)

- SLA of cloud service providers
  - It requires dynamic cloud collaboration management systems which include optimal resource provisioning and load balancing methods to both satisfy QoS and reduce the resource usage cost in heterogeneous cloud collaboration environment.
  - It requires multi-cloud resource brokers which manage allocated virtual clusters or virtual machine instances.
  - It requires cloud node adaptors which provide resource provision by access to distributed heterogeneous cloud resource.
  - It requires job workflow managers which divide applications into sub jobs, allocate the jobs to distributed cloud servers using resource in cloud collaboration environment.
  - It requires monitoring interfaces which monitor cloud resource status.
  - It requires resource description parsers which parse client's requests as the certain format.
  - It requires ACCP cloud manager interfaces which profile services and allocate virtual machine instances.
  - It requires resource provisioning managers which operate effective resource provisioning to provide optimal resource to clients and prepare resource in reasonable cost, and provide load balancing to distribute load.

1.2.3 Application Characteristics and Requirements for Application Aware Resource Management Scheme in Heterogeneous Cloud Environment

Application profiling is a technique used to describe the use of computing resources by an application and its expected behaviors. It should be used by cloud providers to better understand and manage applications and resources. Considering the growth of cloud computing and direct resource utilization impacts to costs, to perform application profiling is essential with these three main reasons.

Application management—environments in which applications share resources have to predict needs for resources properly. In this way, it can be allocated with the amount needed for each workload to perform its job as expected by its end users. Therefore, in order to estimate the amount of resources that should be allocated, an accurate tool is needed to predict applications' need, so as to prevent service degradation generated by resource contention that occurs when applications compete for resources.

Resource management—in order to optimize resource utilization, a model is essential to predict the amount of resources that best suits each workload. Enabling cloud providers to consolidate workloads while maintaining SLAs.

Cost management—in a cloud environment the costs are directly bound to the amount of resource used to provide an application/service. Therefore, using accurate models is possible to consolidate workloads and thus cause little or no impact on application performance while reducing the costs with management and provisioning.

- Application characteristics

The workflow applications executed in cloud environment are classified as data-intensive application, compute-intensive application and instance-intensive application. Especially, the workflow applications for science field are classified as data-intensive applications and compute-intensive applications. The data-intensive applications require the data file capacity from Giga-Byte to Peta-Byte which is higher than the computation workload. The compute-intensive applications require more computation workloads than data workloads. Computation to Communication Ratio (CCR) is used to distinguish between data-intensive application and compute-intensive application. If CCR is lower, the application is closed to the data-intensive application. The instance-intensive applications require many instances in parallel but not high capacity on each instance. Most of applications are compute-intensive applications and have the workflow type which has the order and dependency between tasks.

- Profiling characteristics

The creation of an application profiling involves collection, processing and analysis of different data sets. These sets can be traces of resource usage, such as CPU, memory, network bandwidth or metrics related to provide applications/

services such as number of requests that is being served, the application's architecture, etc. A model that represents the state of the art of cloud application profiling, forecasting and management should have the following characteristics:

- Accuracy—when traces of resource usage is collected to create a historic database, the tool/model used to collect the data should be accurate, not just to count the amount of resources that is being used directly by an application, but also the amount of resources used to manage the application itself. That extra resource should be taken into consideration by a profile, management and forecasting model. Hence, physical nodes that may be elected to host the workload need to have the amount of resources needed by the application plus the amount required to manage it.
- Application design—today, most of applications are developed to bind different components together, such as database server, application server and front-end server. When the application is deployed in a cloud, each component is normally configured into distinct virtual machines. This way, if we scale up the amount of resources available for one of those components to avoid service degradation due to a sudden increase on requests, it is also needed to scale up the amount of resources proportionally on the interdependent layers. Otherwise, we solve a problem in one of the layers pushing it to a dependent one.
- Background workload—it is necessary to monitor the background workload that the physical host has, in order to identify interferences that one application can have on another. Thus, enabling the identification of incompatible applications, which cannot share the same physical server, may lead to competition for the same resource among them.
- Historic data—it is essential to collect and store resource usage traces. Every single resource that could affect the application behavior should be monitored and stored. These traces can be used to detect patterns of workloads that may arrive over the time.
- Migration—monitoring future resource needs is vital. Because it is possible to check whether a physical host is running out of resources in time to activate the migration process before the server gets flooded and the application suffers from service degradation. The migration process has high computing costs, therefore, it should be triggered before the server gets flooded, otherwise applications may suffer from resource contention caused not just by applications competing for resource, but also by resource contention generated by the migration process itself.
- Networking affinity—workloads deployed in different hosts communicate with each other using physical networking structures. Therefore, workloads with frequent communicate should be placed into the same physical node or in the nearest one, in order to avoid networking hops. Hence, applications may suffer service degradation due to flooded network. Moreover, multi-tier applications could benefit from this characteristic, speeding up the communication between layers, hence their packages will be exchanged in memory.

- Overhead—the application profiling models need to constantly monitor a variety of application characteristics and physical servers. It also needs to process and analyze those data in a real time manner to support the cloud management processes. Hence, those models should be aware of the overhead that is caused by them, and try to minimize the impact that it has on services provided by the Cloud.
- Request types—it is related to the collection and classification of request types that the application is serving to future correlation and analysis with traces of computing usage. In this way, we can identify different request groups ranging from most sensitive ones that need priority to the ones that can suffer some time delay that servers are loaded.
- SLA—by constantly monitoring the SLA, providers have means to tune up the amount of resources allocated to workloads. SLAs are the guidelines to be followed toward QoS assurance. Moreover, there is quality of experience (QoE) which is the behavior perceived by end users. Therefore, providers should carefully manage SLAs.

1.2.4 Architecture of Brokers for Heterogeneous Cloud Resource Management

According to Gartner, cloud service brokers (CSBs as shown in Fig. 1.6) are one of the top 10 strategic technology trends in 2014 [10]. There are many companies to

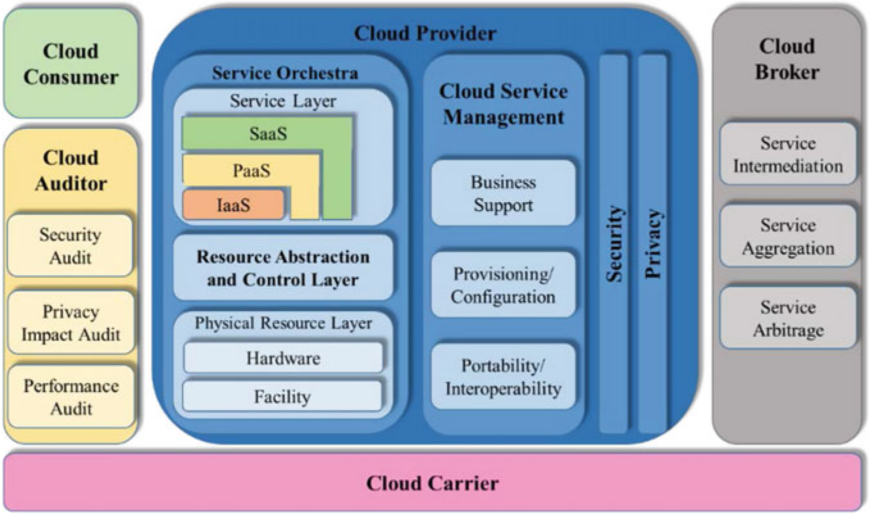


Fig. 1.6 A conceptual reference model of cloud service brokers in NIST [17]

serve CSBs and their roles are mainly selecting the best services of multiple clouds, adding monitoring services and managing metadata services, and providing Software-as-a-Service (SaaS). Liu et al. [11] classified these services as three forms: service intermediation to improve services by adding new value-added features, service aggregation to combine and integrate services into new services, and service arbitrage to arbitrage and aggregate service with not fixed services. In addition, there are many variations of those.

A reservation-based cloud service broker (R-CSB) [12] executes applications on behalf of CSCs (Cloud Service Customer) or provides SaaS using VMs leased from CSPs (Cloud Service Provider). A profit of the R-CSB is made by an arbitrage between CSCs and CSPs, and service fees from CSCs. To increase the profit, the VM leasing cost of the R-CSB should decrease and we solve it via cost-effective VM reservation and allocation. The VM reservation is based on the following facts. The resources provided by CSPs is generally divided by OVMs (On-demand Virtual Machine) and RVMs (Reserved Virtual Machine). The OVMs and the RVMs refers to VMs which are leased in comparatively short BTUs (Billing Time Units, e.g. an hour) and long BTUs (e.g. a month, a year), respectively. Prices of RVMs per unit time is set to be cheaper than those of OVMs and the VM reservation can reduce the VM leasing cost. However, because BTUs of RVMs are much longer than those of OVMs, the VM leasing cost can rather increase if utilizations of the RVMs are low. Therefore, the R-CSB should lease an appropriate number of RVMs. Moreover, the VM allocation decreases the VM leasing cost via increasing average VM utilization. Generally, demands vary by time. If the number of leased RVMs is greater than or equal to the current demand, it is sufficient to allocate applications to them and the OVM leasing cost is not imposed. Otherwise, an additional OVM should be leased to allocate the application. Therefore, increasing average VM utilization decreases the number of OVMs and results in decrease of the VM leasing cost.

A VM reservation module is to determine the number of RVMs to be leased by time. The VM reservation strategizing in the VM reservation module is designed to perform based on demand monitoring and prediction. RVMs leased by the VM reservation module and OVMs additionally leased are managed in a VM pool management module and used to allocate applications. We divide the VM pool into two kinds: VM pools which contains VMs whose status are idle (an idle VM pool) and VMs on which the applications are executed (an active VM pool). For application execution requests of CSCs via a user interface, the R-CSB parses the application execution requests and profiles the applications if the profiling isn't done before. The applications are scheduled and allocated to appropriate VMs in the idle VM pool and VM scaling is performed if it is empty. Then, the application execution module starts to execute the applications via a cloud interface.

1.3 Adaptive Resource Collaboration Framework [13]

In Cloud computing, multiple CSPs provide their own services with various performance and cost. Cloud service user should integrate a number of services from different CSPs and utilize proper services considering cost, performance, and target application's characteristics. Also, active cloud collaboration is needed. Therefore, integration technique for integrating cloud-based resources and cloud collaboration technique is essential.

We consider cloud broker platform prior to development such techniques. Cloud broker plays a role as a middleware between multiple heterogeneous cloud environment and cloud service user. It determines and provides proper cloud services which ensures user SLA and minimize the cost instead of cloud user.

However, cloud broker for federated cloud which have heterogeneous OS, heterogeneous cloud platform, and various cloud service resources only performs integration of multiple CSPs and requests VM instance. Recent cloud broker also provides only the function of VM instance specification and resource find operation (Fig. 1.7).

In cloud collaboration environment (Fig. 1.8), VMs with the same flavor type may have different performance because of heterogeneity of physical nodes and VM interference. However, traditional cloud broker cannot cope with that situation so that SLA cannot be ensured.

Also, in the case of guaranteeing network performance between VM instances are important, traditional cloud broker do not consider this situation so that they can't assure SLA. Also, traditional cloud broker only considers to provide on-demand VM instances so that they do not consider cost reduction using reserved VM instance.

A cloud broker platform named Cloud Collaboration Platform(CCP) presented in Fig. 1.9 [13] considers computing, network performance using cloud resource profiling function. Also, CCP reduces VM management cost by adapting reserved VM instances and VM provisioning operation so that it dynamically provides optimized service while meeting user SLA and QoS with minimized cost.

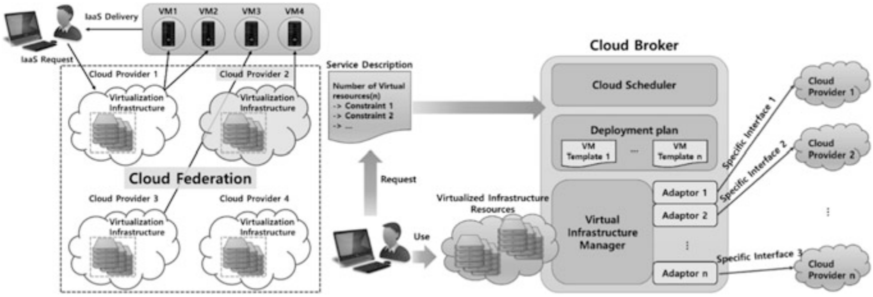


Fig. 1.7 Traditional cloud broker models

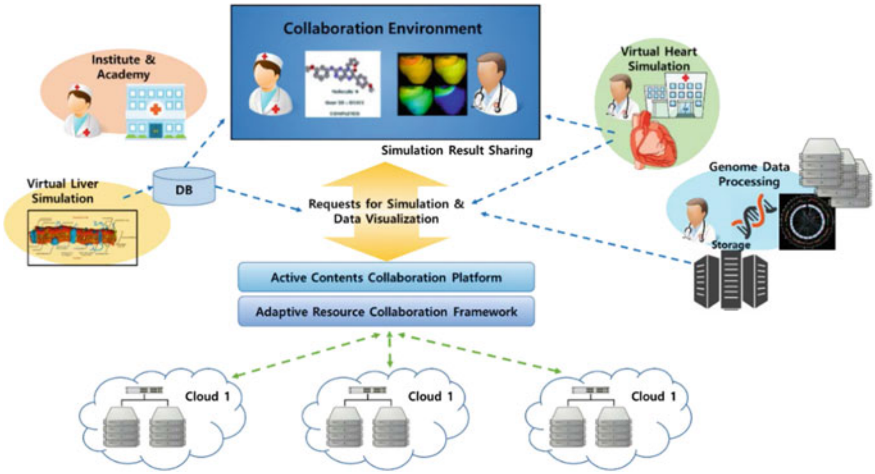

Collaboration Environment
Institute & Academy
Virtual Heart Simulation
Simulation Result Sharing
Virtual Liver Simulation
DB
Requests for Simulation & Data Visualization
Genome Data Processing
Storage
Active Contents Collaboration Platform
Adaptive Resource Collaboration Framework
Cloud 1
Cloud 1
Cloud 1

Fig. 1.8 Cloud collaboration environment [13]

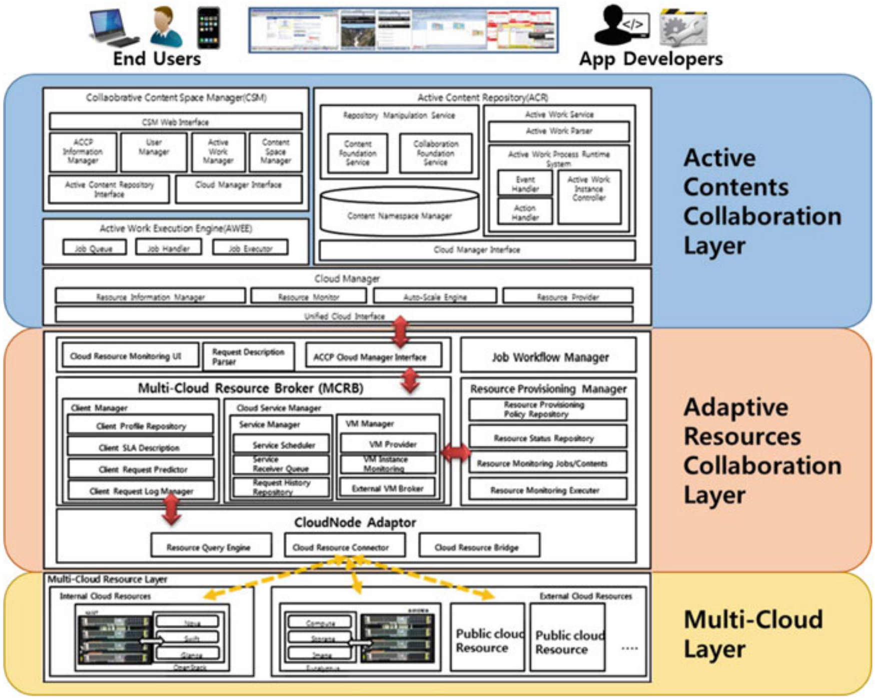


Fig. 1.9 Cloud collaboration framework [13]

Moreover, in cloud environment, many requests are in a form of running scientific applications or workflows. Therefore, CCP provides workflow scheduling function to optimized use of resources. For contents management, CCP manages multi-tenant and provides interface which is suitable for active collaboration.

The entire framework for CCP is divided into Active Contents Collaboration Platform (ACCP) and Adaptive Resource Collaboration Framework (ARCF) [13].

ARCP manages heterogeneous multi-clouds in an integrated form and provides proper cloud resource in cost adaptive way while meeting user QoS. ARCF provides policy based resource management mechanism considering multiple cloud services' performance and cost. Moreover, ARCP provides application service for various scientific applications such as genetic analytics, scientific experiments.

ACCP provides collaboration environment for sharing services and contents in a virtual workspace based on cloud. Also, it automatically performs active work according to the event occurred by contents. ACCP provides scalable service through interconnection to cloud infrastructure, and development environment for collaborative works. Also, ACCP provides virtual workspace named content space so that users can rapidly and easily collaborate with co-workers with it.

It provides Content Space, which operates as the interface for task flow processing service to offer a convenient environment for various collaborative works.

Cloud collaboration technique provides contents collaboration environments required for next-generation mobile services such as business and social media. Also, it integrates different services from diverse cloud vendors then provides a unified interface so that users can easily develop and quickly utilize an application. In this document, we concentrate upon taking care of Adaptive Resource Collaboration Framework.

1.3.1 The Architecture of ARCF [13]

Cloud resource management architecture consists of user, Multi-cloud Resource Broker (MCRB), Cloud Node Adaptor (CNA) and Workflow Manager(WM). A user can submit service requests regardless of geographical location. MCRB serves a role of interface between cloud service provider and user. WM and MCRB decides whether to accept the submitted request. If it is accepted, then they choose proper resource to allocate the request then schedule and execute it.

We define ARCF (Fig. 1.10) and list the modules which are the parts of the framework as following:

– Multi Cloud Service and Resource Broker (MCSRB): It analyzes resource request and finds the most proper resource to satisfy SLA.
– ARCF Adaptor: External user can utilize VM instances through this interface. Also, it is possible for a user to see collected VM instance monitoring information.

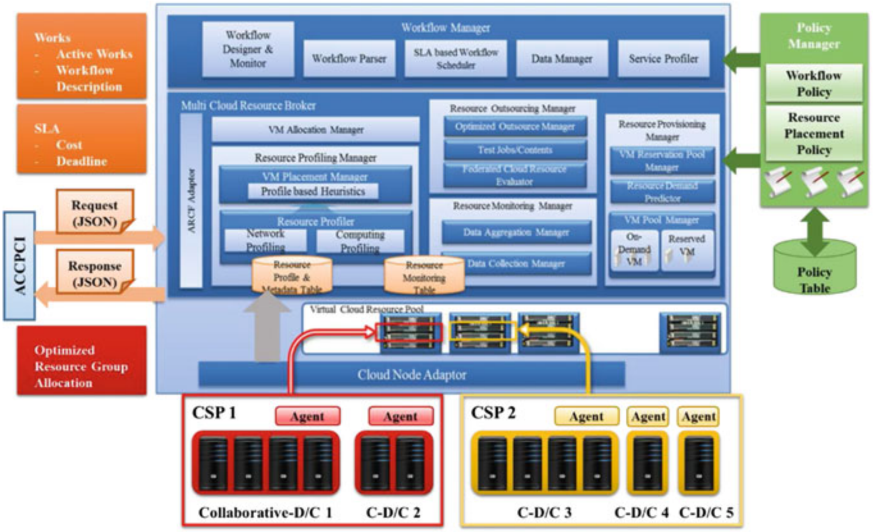


Fig. 1.10 ARCF architecture [13]

- Virtual resource allocation manager: It predicts future client request based on client request history information collected by request log manager. Request log manager collects client profile information, SLA information and requirement information.
- Resource profiling and monitoring manager: It is composed of VM manager and service manager. It assigns proper VM instance to the request according to the request analysis. After assigning VM instances, it monitors additional information such as task completion time. Also, it evaluates SLA for submitted requests. After evaluation, requests are given priority depending on SLA. According to the priorities, requests are matched with proper resources and processed.
- Resource Provisioning Manager (RPM): It utilizes resource provisioning and balancing scheme to process user request efficiently. Also, it allocates the request onto the chosen resource by CloudNode Adaptor which handle heterogeneous cloud resources. Also, it employs the most decent policy among defined resource provisioning policy to process the request economically.
- CloudNode Adaptor: It delivers heterogeneous cloud services which are provided by different cloud vendors. Also, it integrates cloud resources with heterogeneous OS and heterogeneous cloud platform environment with Cloud Resource Bridge module. Cloud Resource Connector module, which is a part of CloudNode Adaptor, receives messages from cloud service providers.
- Job Workflow Manager (JWM): It helps executing high performance computing application (e.g. Next-generation sequencing application).

First essential functionality of ARCF takes care of VM placement problem. It equips with VM placement technique with two QoS constraints that client cannot handle. One of them is guaranteeing network performance among multiple VM instance. We provide network-aware VM placement scheme to overcome data transmission delay. The other is computing aware VM placement scheme. It enables for a client to use VMs of same computing power with cheaper price by performing resource profiling.

Second functionality is resource monitoring. It periodically monitors ARCF-governed VM instances to collect resource-related information such as task execution time.

Third functionality is resource provisioning. We considered reserved VM to reduce resource leasing cost. We achieve this goal by employing two techniques; One of them is Adaptive Resource Reservation Scheme (ARRS). It decides optimized number of newly leasing reserved VM by leveraging the concept of marginal cost. The other is Adaptive Resource Allocation scheme. It consists of recycling reusable on-demand VM (OVM), replacing targeted VM type, and repositioning the task in execution from OVM to RVM.

Fourth functionality is workflow scheduling. It is about processing active works while satisfying user SLA (e.g. deadline). When multiple workflow processing request is given, we partition each workflow request into fragment based on its unique critical path and schedule each fragment. Also, as cloud resource performance is not static, we suggest a scheduling scheme which divides a task into multiple subtasks then process them in parallel manner to overcome the unforeseen performance degradation.

1.3.2 Resource Monitoring

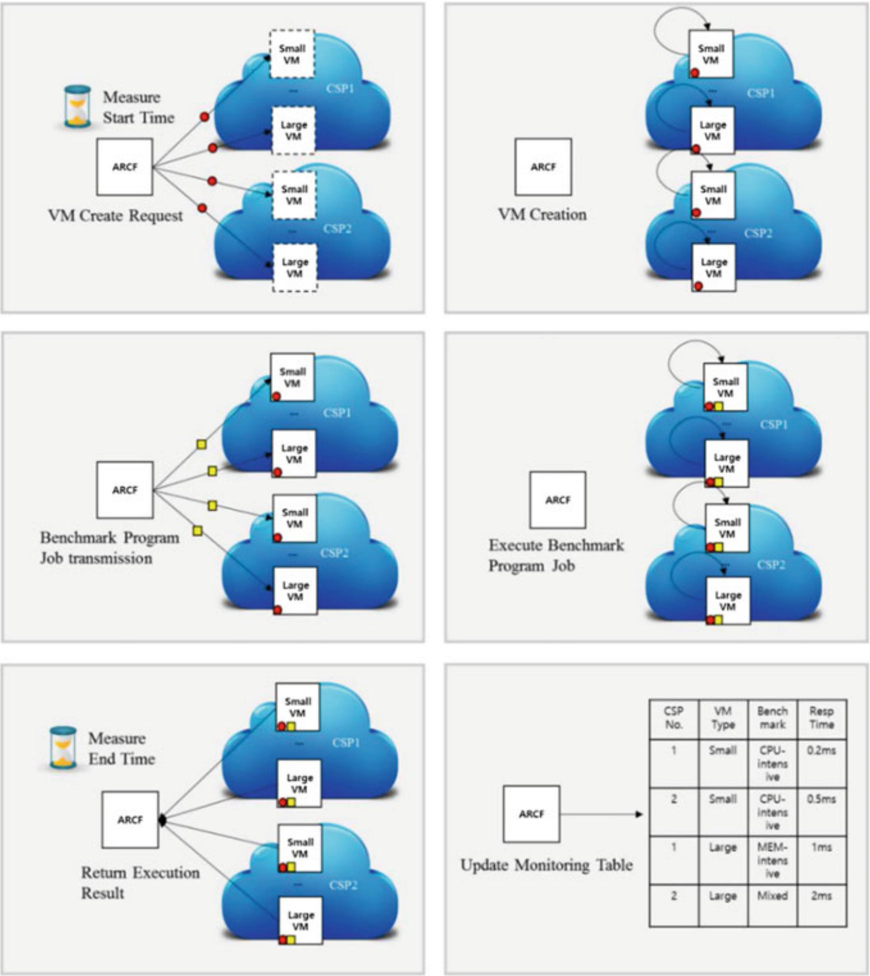
Resource monitoring basically collects and maintains the periodic performance information and the status of created cloud resources in VM pool or resource policy provided by CSPs. The scalability of monitoring technique should be guaranteed not to exceed the threshold of the performance degradation overhead even when the number of cloud resource monitored is increased.

Resource monitoring provides two types of monitoring. The first type provides the real time monitoring for performance evaluation of resource policies using benchmark program. The second type provides the real time monitoring for checking the status of the created cloud resources in VM pool using daemon process.

(1) **Monitoring using benchmark program**

To evaluate the performance of resource policies provided by CSPs, resource monitor provides the periodic monitoring using benchmark program from benchmark repository. Figure 1.11 shows the monitoring process using benchmark program in resource monitoring.

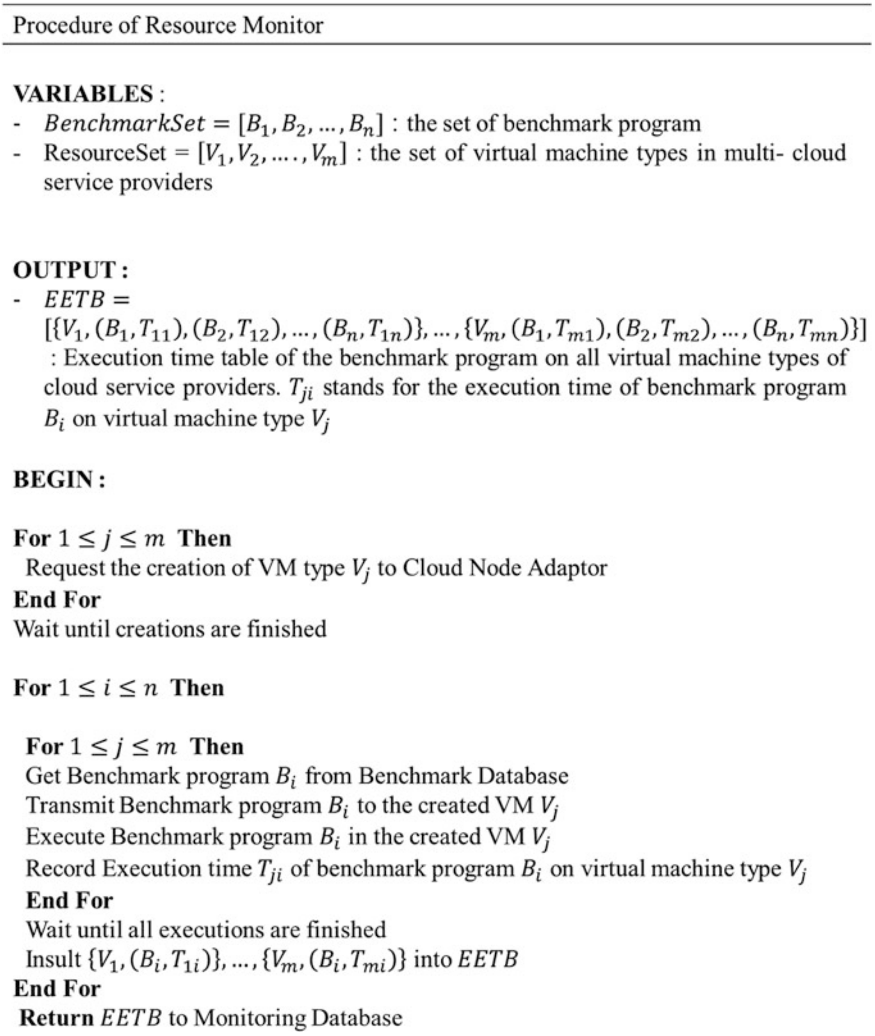
The administrator of integrated broker registers various benchmark programs to benchmark repository in advance and the benchmark repository provides the VM



CSP No.	VM Type	Bench mark	Resp Time
1	Small	CPU-intensive	0.2ms
2	Small	CPU-intensive	0.5ms
1	Large	MEM-intensive	1ms
2	Large	Mixed	2ms

Fig. 1.11 The monitoring process using benchmark program [13]

images of each CSP installing benchmark programs. Resource monitor broadcasts the request to create VM on available resource policies such as small type, medium type or large type which provide CPSs with this VM image. After completing to create the requested VMs, resource monitor broadcasts the request to execute each benchmark program on the requested VMs. After finishing the execution of the benchmark program on the requested VMs, the result in terms of the execution time of the benchmark program is returned and monitoring database is updated. Based on this procedure shown in Fig. 1.12, resource monitoring can evaluate the performance of resource policies provided by CSPs with various benchmark programs. This monitoring data will be used by resource profiles.



```
Procedure of Resource Monitor

VARIABLES :
- $BenchmarkSet = [B_1, B_2, ..., B_n]$ : the set of benchmark program
- ResourceSet = $[V_1, V_2, ...., V_m]$ : the set of virtual machine types in multi- cloud
  service providers

OUTPUT :
- $EETB =$
  $[\{V_1, (B_1, T_{11}), (B_2, T_{12}), ..., (B_n, T_{1n})\}, ..., \{V_m, (B_1, T_{m1}), (B_2, T_{m2}), ..., (B_n, T_{mn})\}]$
   : Execution time table of the benchmark program on all virtual machine types of
  cloud service providers. $T_{ji}$ stands for the execution time of benchmark program
  $B_i$ on virtual machine type $V_j$

BEGIN :

For $1 \le j \le m$ Then
  Request the creation of VM type $V_j$ to Cloud Node Adaptor
End For
Wait until creations are finished

For $1 \le i \le n$ Then

  For $1 \le j \le m$ Then
  Get Benchmark program $B_i$ from Benchmark Database
  Transmit Benchmark program $B_i$ to the created VM $V_j$
  Execute Benchmark program $B_i$ in the created VM $V_j$
  Record Execution time $T_{ji}$ of benchmark program $B_i$ on virtual machine type $V_j$
  End For
  Wait until all executions are finished
  Insult $\{V_1, (B_i, T_{1i})\}, ..., \{V_m, (B_i, T_{mi})\}$ into $EETB$
End For
 Return $EETB$ to Monitoring Database
```

Fig. 1.12 The procedure of resource monitor [13]

1.4 Science Gateway Overview

Many computational science field (i.e. Computational Physics, Computational Chemistry, Bioinformatics and etc.) related to big-data domain [14, 15] require the massive computation resources to solve their problems with methods of simulation, modeling and numerical analysis.

As a substitute for traditional expensive supercomputing, the cloud computing which can deliver cost efficient and scalable computing services which are provided to cloud users on-demand is becoming an alternative computing paradigm for scientific application processing. Cloud users are able to share computing resources such as computing components (i.e. CPU, memory and network), software utilities, databases, etc. for scientific application processing cost-effectively by leasing virtual machine (VM) instances on a pay as you go basis. Despite its many advantages, there are several obstacles which prevent the cloud computing from achieving successful deployment for scientific application processing.

Advanced scientific applications have generally diversified computing requirements such as CPU intensive, memory intensive and network intensive computation and so on [16, 17]. Therefore, it is difficult to derive a resource scheduling scheme which is a panacea for all kind of scientific applications. Moreover, we may observe that there is a conflict among the scheduling objectives such as application completion time and resource cost [18, 19]. The improvement of one objective might bring down the performance of other objective. In addition, the professionals engaged in computational science might suffer from the complexity of distributed computing system (e.g. heterogeneous resource management, pipeline flow control, service deployment, intermediate data control, and etc.) in cloud for solving their problems. To overcome above issues in order to achieve the performance and cost efficient resource scheduling for scientific application processing on cloud, the concept of science gateway is proposed.

A science gateway is an automatic execution environment that can be generally applied for different types of science applications in common interface to do task scheduling, task execution and visualization, building highly tuned computation farm over the geographically distributed resources [20]. The science gateway cloud supports the generalized common interface which is available regardless of the kind of requested scientific application. Even cloud users who are not familiar with the traditional cloud scientific systems can optimize the scientific application processing performance simply without in-depth understanding of cloud platform. All they need is just to provide the description on a set of tasks with precedence constraints including in their scientific applications. The important issue of the science gateway is to orchestrate tasks over a distributed set of resources in the minimized cost while guaranteeing SLA which is the contract on QoS constraints with a user.

1.5 Scientific Workflow Applications

Advanced computing application involves the simulation and modeling program to solve the problem in many computational science fields such as computational physics, computational chemistry, bioinformatics, etc. These applications related to the big-data computing [14, 15] require massive computation resources to solve various problems with methods of simulation, modeling and numerical analysis. While generating massive streams of data continuously, enormous parallel or distributed supercomputing could be required in proportion to their computational complexity [16, 17, 21]. Advanced computing applications are computing-intensive, data-intensive and time-consuming and hence, advanced computing applications need a huge size of computing and storage resource [16, 22–25]. In this section, we focus on the scientific applications among advanced computing application service such as computational chemistry tool and bio-computing tool.

1.5.1 Programming Models for Scientific Applications [26]

Scientific applications involve the construction of simulation and modeling techniques to solve scientific or engineering problems. Such the applications practically need a huge size of computing resources and storage space to perform a large set of experimental parameters or to analyze a huge size of dataset. In addition, the different services or applications have the different requirement on resource. For executing the scientific applications, there are various programming models such as Thread Model, MapReduce Model, MPI Model, and Workflow. Table 1.3 shows a feature comparison of these programming models on the applications, scenarios, execution unit, and execution services for supporting the scientific computing. In this book, we adopt the workflow as the programming model to execute the scientific applications so, assume that these scientific applications are represented as workflow which is directed acyclic graph (DAG) composing of the set of node and the set of edge (called scientific workflow).

Table 1.3 Programming Models for Scientific Applications [26]

Name	Scenario	Applications	Execution unit	Execution service
Thread model	Multi-threaded applications	A collection of threads executed concurrently	Any instance, any method	Thread scheduling service and thread execution service
MapReduce model	Data-intensive applications	A map and a reduce functions and a large collection of data	Map and Reduce Tasks	MapReduce scheduling and execution services, MapReduce storage service
Workflow	Workflow applications	A collection of interrelated tasks composing a DAG	Task instance	Built on top of the task model with additional requirements
MPI	Message passing applications	A collection of MPI processes that exchange messages	MPI processes	MPI scheduling service, MPI execution service

Table 1.4 Requirement for Executing Scientific Application on the Cloud Environment [27]

Requirement	Description
SLA based in-time workflow scheduling mechanism	Owing to the dynamic characteristic of distributed resources, an in-time scheduling mechanism for workflow management guaranteeing the SLA required by users is needed
Policy decision mechanism	To manage the policies for different users' intention, a decision scheme is needed to choose the suitable policy for the workflow scheduling based on the modeled SLA
Application profile mechanism	Since the finish time factor is included in SLA, we should have a mechanism to predict the future runtime of a task in our system. The time prediction will be realized through analysis of historical execution data
Cloud resource management mechanism	To allocate the tasks of workflow to the dynamic and heterogeneous cloud environment, a resource management function is required

To execute these scientific workflows on the cloud environment with following features shown in Table 1.4, the science gateway which is the policy based workflow management integrated cloud resource broker system is proposed in [27].

1.5.2 Next Generation Sequencing for Genome Analysis

In biological research and applications to understand biological phenomena and relation between genotype and phenotype, it is essential to determine the order of the nucleotide bases in DNA molecules and analyze the resulting sequences. Next-generation sequencing (NGS) technologies is used to sequence DNA in an automated and high-throughput process [28, 29]. DNA molecules are fragmented into pieces of 100 to 800 bps, and digital versions of DNA fragments are generated accordingly. These fragments, called reads, originate from random positions of DNA molecules. To know the genome sequence of individuals, the aligning or mapping operation is necessary to determine with which location of reference genome reads generated from DNA of individual can match.

Aligning NGS reads to genomes is computationally intensive. Li et al. gave an overview of algorithms and tools currently in use [30]. To align reads containing SNPs which is polymorphism of a single base pairs and is recognized as the main cause of human genetic variability, probabilistic algorithms are required, since finding an exact match between reads and given reference are not sufficient because of polymorphisms and sequencing errors. Most of these algorithms are based on a basic pattern called *seed and extend*, where small matching regions between reads and the reference genome are identified first (seeding), and then further extended. Additionally, to be able to identify seeds that contain SNPs, a dedicated algorithm

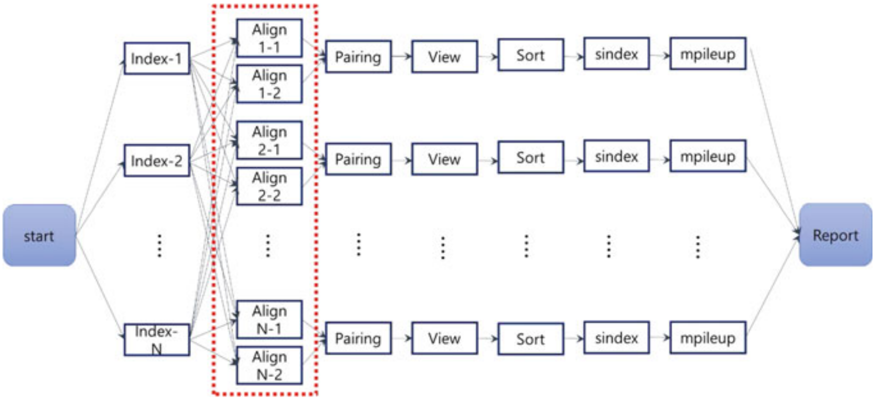


Fig. 1.13 Typical genome sequencing workflow for SNP identification [22]

that allows for a certain difference during seeding is required. Unfortunately, this adaptation further increases the computational complexity.

BWA [24] and SAMtools [31] are the typical bio scientific applications for aligning or mapping low-divergent sequences against a large reference genome, such as the human genome. BWA consists of the services such as bwa index, bwa aln and bwa sampe. Bwa index provides indexing the reference DNA. Bwa aln provides mapping pair between reference-data and sample data. Bwa sampe merges the pair. SAMtools consists of the services such as samtools view, samtools sort, samtools index and samtools mpileup. Samtools view, samtools sort and samtools index provides reformatting. Samtools mpileup extracts the gene variations. These services can be provided by the type of workflow for genome sequencing and Fig. 1.13 show the example of workflow for identifying SNP variations.

Sequencing throughput increases faster and so do the computational power and storage size. As a result, although NGS machines are becoming cheaper, using dedicated compute clusters for read alignment is still a significant investment. Fortunately, even small labs can do the alignment by using cloud resources.

1.6 Conventional Service Broker for Scientific Application in Cloud

There are many distributed solutions to execute these scientific applications [32, 33]. Especially, about large-scale scientific application, recent solutions make use of remote computing farm such as federated clusters or cloud with the objective of reducing the cost by on-demand way instead of maintaining a local computing farm. For example, CloudBLAST [25] shown in Fig. 1.14 provides BLAST service which is a bioinformatics tool to find regions with local similarity between

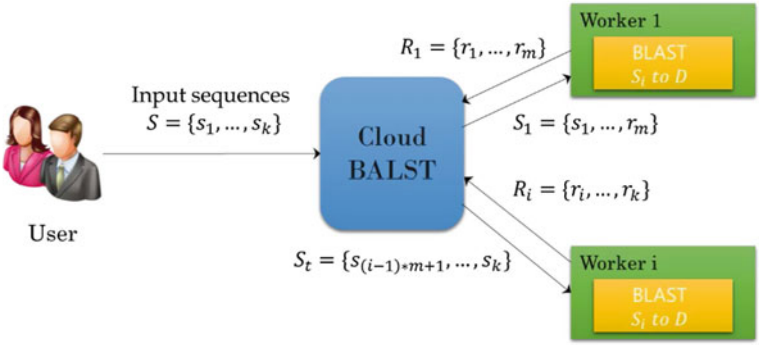

Worker 1
$R_1 = \{r_1, \ldots, r_m\}$
BLAST
S_i to D
Input sequences
$S = \{s_1, \ldots, s_k\}$
Cloud
BALST
$S_1 = \{s_1, \ldots, r_m\}$
$R_i = \{r_i, \ldots, r_k\}$
User
$S_t = \{s_{(i-1)*m+1}, \ldots, s_k\}$
Worker i
BLAST
S_i to D

Fig. 1.14 The processing in BLAST service [25]

nucleotide or protein sequences with parallelization process in Apache Hadoop in cloud environment, so as to control the total execution time.

In the case of these solutions which make use of remote computing farm, the total cost is determined by QoS level such as the processing speed of scientific application services [1]. High QoS can be provided by parallelizing jobs which require many resources; while low QoS can be provided by leasing relative small resources. Therefore, the scheduling and resource management function to parallelize jobs and allocate jobs to distributed resources efficiently is needed in order to reduce the total execution time or cost.

1.6.1 Service Broker for Computational Chemistry Tool

Advanced scientific applications generally have diverse computing requirements, such as CPU intensive, memory intensive, and network intensive computation, and so on [18, 21]. Therefore, it is difficult to derive a resource scheduling scheme which is a panacea for all kind of scientific applications. Moreover, we may observe that there are conflicts among scheduling objectives, such as application completion time and resource cost [19, 20]. The improvement of one objective might bring down the performance of other objectives. In addition, the professionals engaged in computational science might suffer from the complexity of distributed computing system (e.g. heterogeneous resource management, pipeline flow control, service deployment, intermediate data control, and etc.) in the cloud for solving their problems. To overcome the above issues to achieve the optimal performance and cost efficient resource scheduling for scientific application processing on the cloud, the concept of the science gateway is proposed. A science gateway is an automatic execution environment that can be generally applied for different types of science applications in a common interface to do task scheduling, task execution and visualization, building a highly tuned computation farm over geographically distributed resources [34]. The important issue of the science gateway is to orchestrate tasks over a distributed set of resources with minimized costs while guaranteeing SLAs. However, several previous studies on the science gateway have not considered this issue or could not resolve this issue sufficiently. In addition, they focused on a specific scientific application, so they are not suitable for various scientific applications generally.

Particularly, Science Gateway has the following capabilities and properties:

- Have a repository for scientific applications.
- Be interactive with users through interfaces to accept user-composed workflows and SLA specifications
- Be interactive with the service providers through resource management to provision a resource to each workflow sub-task and handle the resource request fluctuation by maintaining a low resource provisioning cost as much as possible
- Can produce the optimal schedule plan for executing the workflow while satisfying SLA, dispatch each task to the selected resource, gather execution results through workflow management
- Have a task execution history repository to store the historical workflow execution information
- Have task monitoring and workflow monitoring to check the status of execution online.

Figure 1.15 shows the layered architecture of Science Gateway.

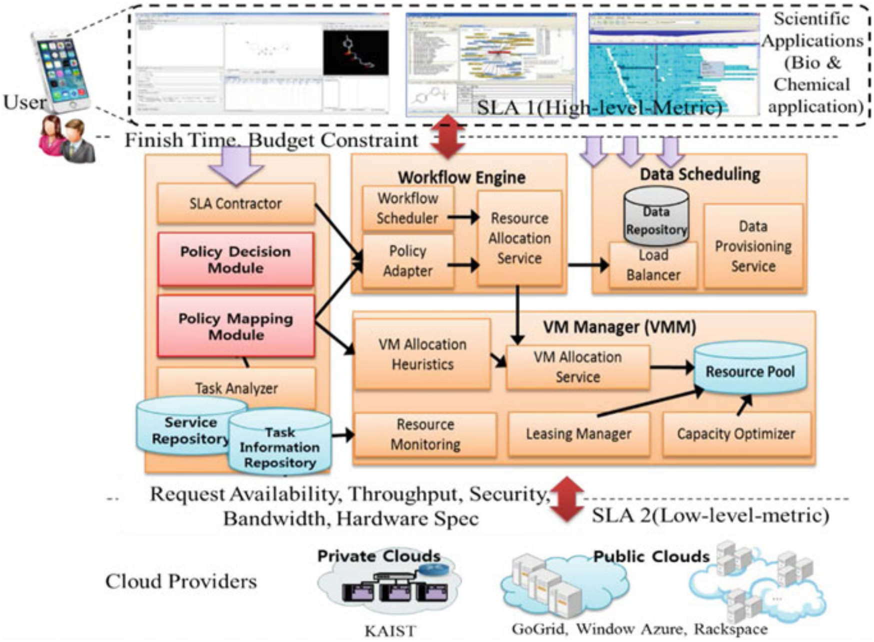


Fig. 1.15 Layered architecture of science gateway [27]

This architecture consists of four layers—chemical service layer, workflow management layer, resource management layer, and cloud resource layer. The brief explanations for the functional blocks and procedure in Science Gateway are as Follows. Users compose their chemistry workflow and submit it with QoS requirements (SLA_1) such as the total execution time and the total budget through web-based unified workflow interface. And workflow manager arranges the QoS parameter (SLA_2) such as cores, memory size and storage size of virtual machine for each sub-task within the workflow based on SLA_1 required by user, according to a chosen policy and historical execution information, and then executes each sub-task of workflow with resource manager. Resource manager then allocates appropriate resources to each sub-task based on scheduling from workflow manager with Cloud Node Adaptor. Auto-scaling scheme is to prepare an appropriate amount of virtual resources in advance in order to reduce virtual resource initiation time delay.

1.6.2 A Distributed Bio-workflow Broker on Clouds

For workflows to cooperate in a real-time manner, Kim in [35] proposed a conceptual idea of the integrated workflow system as shown in Fig. 1.16.

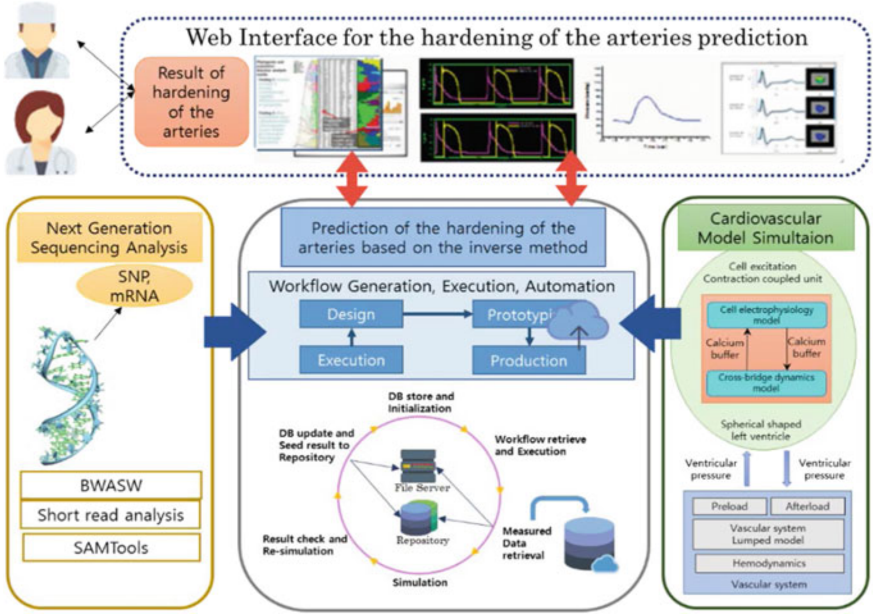

Web Interface for the hardening of the arteries prediction
Result of hardening of the arteries
Next Generation Sequencing Analysis
SNP, mRNA
BWASW
Short read analysis
SAMTools
Prediction of the hardening of the arteries based on the inverse method
Workflow Generation, Execution, Automation
Design
Prototypi
Execution
Production
DB store and Initialization
DB update and Seed result to Repository
Workflow retrieve and Execution
File Server
Repository
Result check and Re-simulation
Measured Data retrieval
Simulation
Cardiovascular Model Simultaion
Cell excitation Contraction coupled unit
Cell electrophysiology model
Calcium buffer
Calcium buffer
Cross-bridge dynamics model
Spherical shaped left ventricle
Ventricular pressure
Ventricular pressure
Preload
Afterload
Vascular system Lumped model
Hemodynamics
Vascular system

Fig. 1.16 System architecture for integrated genome analysis and cardiovascular simulation workflow [35]

The middle layer includes functions which manage the interactions among different bio-workflow services, such as the pressure wave monitoring, the cardiovascular model simulation, and the genome sequence alignment services implemented in workflow model. The system could automate the simulation data and experiment workflow management. Such large-scale data analysis workflow model needs a huge size of computing and storage infrastructure for performing the overall workflow tasks in-house method. However, it is expensive to prepare enough resources and on the other hand, even though it is possible to provide enough resources, the efficiency of the resource utilization is relatively low since all tasks do not require the same computing capacity. On the hybrid cloud model, it is possible to outsource the entire or a part of the workflow tasks into cloud model for workflow computing model which has a distributed workflow services combined by cloud service model.

As shown in Fig. 1.17, a distributed Bio-workflow Broker (DBB) system is located in the middle layer between the end user and the cloud service. The DBB functions as a bridge between the bio services and the cloud data. It stores the bio services such as SNP analysis for DNA or metabolic disease identification system with both of genome database and cell metabolism measurement with unified interface.

To control the workflow execution flow, a task monitoring function is built to identify the task status including submission, execution and publish. However, the solutions focus on static resource provisioning with batch processing scheme in local computing farm and data storage. Since all dataset should be located in the local site before the processing starts, the transfer time of huge dataset as well as the unbalanced execution time of different problem size increase the total completion time. Therefore, an adaptive resource provisioning scheme for both data preparation process as well as data processing process is proposed in [35].

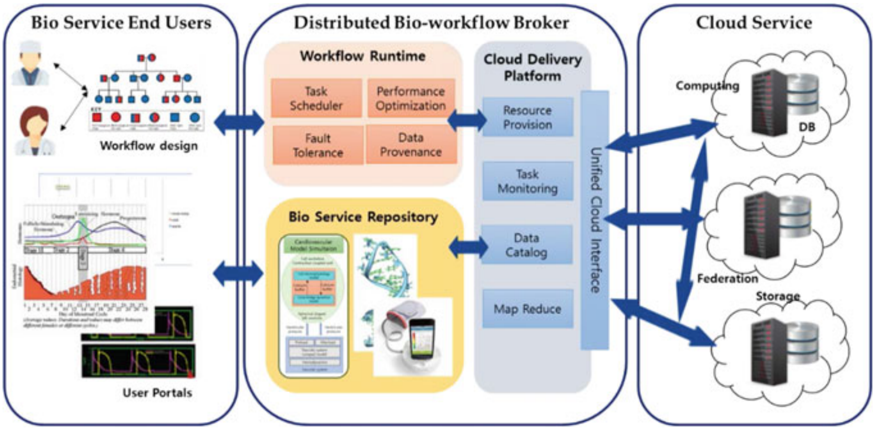


Fig. 1.17 Distributed bio-workflow model combined by cloud service [35]

1.7 Cost Adaptive Resource Management in Science Gateway

For scientific application processing, it is important to reduce the resource leasing cost while guaranteeing the SLA of user's request. In perspective of the science gateway, the one of efficient ways to reduce the resource leasing cost (i.e., maximizing profit) is considering the payment plans of cloud resource providers: reserved VM (RVM) and on-demand VM (OVM) plans.

1.7.1 Pricing Model for Scientific Computing

In the science gateway, there are three entities: the science gateway cloud, cloud service users, and the cloud resource providers. The users submit their scientific applications and data to the science gateway; the science gateway buys or releases cloud resources from/to the cloud resource provider when as needed; and then the science gateway will pick an appropriate VM instance from its own VM pool to execute each task of the scientific applications. The cloud service users pay the science gateway cloud for their application processing requests. The science gateway cloud makes a profit while satisfying user's requirements as a wholesaler between cloud service users and cloud resource providers. Generally, scientific applications have precedence constraints. Therefore, each task is allocated to their suitable VM instances and executed in order of their starting time based on precedence constraints. Obviously, each task has different performance on different types of VM instances. Scientific applications may have different levels of importance and urgency; therefore, users can specify different deadlines for their applications. Since the profit of the science gateway is largely dependent on the arrival density and the structure of scientific application requests, the adaptive pricing model for requests is important to maximize the profit (Table 1.5).

Table 1.5 Variables for science gateway cloud [27]

Notations	Descriptions
t_i	A ith task of scientific application S $i \in \{1, 2, \ldots, N\}, N = \mathit{number\ of\ tasks}$ in S
VT_i	ith VM instance type $VT \in \{1, 2, \ldots, K\}$
$R_i(\tau)$	The number of i-type available RVM instances in VM pool at time τ
$A_i(\tau)$	The number of i-type allocated RVM instances in VM pool at time τ
$N_i(\tau)$	The number of i-type leased RVM instances in VM pool from cloud resource provider at time t
$R_{i,j}$	The jth RVM instance of i-type, $\sum_j R_{i,j}(\tau) = N_i(\tau)$
C_i^{max}	The maximum price of i-type VMs to users
C_i^{min}	The minimum price of i-type VMs to users
$C(R_{i,j}(\tau))$	The current price of the jth RVM instance of i-type at time τ

Definition 1.1 (*SLA constrained scientific application*) From the users' perspective, they hope the scientific application they submitted can be finished within some specified deadline or budget. For example, if users will specify deadline constraint D, that is to say, the user want to run his or her scientific application no later than a specified deadline. Meanwhile, the user hopes that the scientific application should be finished with the least possible cost. In such a case, the scientific application can be described as a tuple $S(\gamma, \Phi, D)$ where γ is the finite set of tasks $t_i(i \in \{1, 2, \ldots, N\})$ and Φ is the set of directed edges of the form $\left(t_{i,}t_j\right)$.

For an arbitrary precedence constrained scientific application $S(\gamma, \Phi, D)$, we assume that we are able to know the estimated completion time of each task of S, then we can obtain the optimized resource management policy of the science gateway for scientific application processing which satisfies following objective function.

$$\begin{aligned} & Maximize\, P[\rho] = \sum_{i \in I} c_{sal}(\rho) \cdot (\bar{r}(S_i(\gamma, \Phi, D)) - \delta) - c_{\exp} \sum_{i \in I} r(S_i(\gamma, \Phi, D)) \\ & Subject\, to\, ECT[S_i] = max_{\forall t_{i,j} \in S_i} \left\{ft\left[t_{i,j}\right]\right\}, \quad \forall i \\ & ECT[S_i] \leq D_i, \quad \forall i \end{aligned} \tag{1.1}$$

where $P[\rho]$ is a profit function of the science gateway with input parameter ρ which is a policy for determining a price of service sales to user, c_{sal}. $c_{exp}\left(c_{sal} \geq c_{exp}\right)$ is an expenditure for processing scientific application. δ is a coefficient and $c_{sal} \cdot \delta$ means a preference degradation level of service purchasing from the science gateway. $\bar{r}$ is an expected resource requirement in view of cloud service user for scientific application request $S(\gamma, \Phi, D)$ and r is an actual resource requirement in view of science gateway cloud based on its VM pool. *ECT* is an estimated completion time of scientific application and *ft* is an estimated finishing time of individual task. $t_{i,j}$ is a *j*th task of application S_i. Equality constraint of Eq. (1.1) means the finishing time of last task is the completion time of application.

The cloud resources are categorized into several types, such as small, medium, large, xlarge. Each type of VM instance offers different processing capacity and they are charged for usage in Billing Time Unit (BTU) in proportion to their capacity. Partial-BTU consumption is rounded up to one BTU.

Definition 1.2 *(Cloud VM resource type) A VM type* VT_i can be modeled with four parameters which are time-invariant and continuously capable of being guaranteed by cloud service providers: number of compute units (can be transferred into MIPS) VT_{c_i}, CPU clock rate (Hz) VT_{hz_i}, memory size (GBs) VT_{m_i}, storage space (GBs) VT_{s_i}, and bandwidth (bit rate, bit/sec) VT_{n_i}. The tuple that represent a VM instance: $VT_i = \{VT_{c_i}, VT_{hz_i}, VT_{m_i}, VT_{s_i}, VT_{n_i}\}$.

In general, the cloud providers have two VM instance payment plans such as reserved VM (RVM) and on-demand VM (OVM) plans [36, 37]. For RVM plan, VM instance is leased for long BTU (e.g., monthly or yearly) with low price per allocation time unit. On the contrary, for OVM plan, VM instance is allocated for short BTU (e.g., hourly or daily) with high price per allocation time unit. The science gateway maintains the certain number of RVMs in its own VM pool and adjusts the number of them when the amount of requests is drastically changed. Obviously it is reasonable to prefer RVM to allocate task when we can find available RVMs in VM pool since the cost for OVM is more expensive than the one for RVM.

In order to decide the proper cost of service sales c_{sal} in Eq. (1.1) to maximize the profit of the science gateway, we propose three cost function models: simple, linear and exponential cost function model. The cost for i-type VM instance is between the maximum cost C_i^{max} and C_i^{min} and determined according to the number of available RVMs at each period.

Simple cost function [17] The current price is not changed when available VMs exist as the minimum cost. However, when all the VMs are allocated, the price jumps to the maximum cost as follows,

$$C(R_i(\tau))_{simple} = \begin{cases} C_i^{min} & R_i(\tau) > 0 \\ C_i^{max} & otherwise \end{cases} \tag{1.2}$$

Linear cost function [17] The current price is increased linearly when the number of available RVMs is decreased until the price reaches to the maximum cost as follows,

$$C(R_i(t))_{linear} = \begin{cases} C_i^{min} & R_i(\tau) = N_i(\tau) \\ C_i^{min} + R_i(t)\frac{C_i^{max} - C_i^{min}}{N_i(\tau)} & 0 < R_i(\tau) < N_i\,(\tau) \\ C_i^{max} & R_i(\tau) = 0 \end{cases} \tag{1.3}$$

Exponential cost function [17] The current price is increased exponentially when the number of available RVMs is decreased until the price reaches to the maximum cost as follows,

$$C(R_i(\tau))_{exp} = \begin{cases} C_i^{min} & R_i(\tau) = N_i(\tau) \\ C_i^{min} \cdot exp\left\{\frac{1}{N_i(\tau)} ln\left\{\frac{C_i^{max}}{C_i^{min}}\right\} R_i(\tau)\right\} & 0 < R_i(\tau) < N_i \\ C_i^{max} & R_i(\tau) = 0 \end{cases} \tag{1.4}$$

Consequently, the science gateway is able to optimize its profit by using the proper cost function.

1.7.2 Cost Adaptive Resource Allocation in Science Gateway

The VM pool manager (VMPM) and its policies can be described in terms of reducing the resource leasing cost. Especially, the VMPM determines the proper amount of leasing RVMs from the cloud resource provider to optimize the cost efficiency. The amount of running RVMs is dependent on the arrival request density of scientific applications and each usage duration.

Algorithm 2 shows VMPM resource allocation policy to reduce the cost by using RVM leasing. The historical data including all the executed tasks and their allocated VM instance types during the previous time interval T' is the input data for Algorithm 1. We assume that the request pattern in the current time interval T will be same to the one of historical data in $T^{'}$. By using Algorithm 1, we can derive the proper amount of RVM, N_i for T.

From line 01–04 in Algorithm 2, we first do clustering each task in S according to their allocated VM instance type VT. Consequently, all the tasks in S are classified into several clusters Cls_{VT} according to VT. From line 06–13, we make groups in which have batch of non-overlapped tasks. That is, from the arbitrary Cls_{VT}, the first task is picked if its start time st is later than the finish time ft of last task in the group g_n, and then it is inserted into g_n. This procedure is repeated until we cannot find the available t in Cls_{VT} more. From line 14 to 20, by using group completion time of g_n, $gct(g_n)$ and allocated VM instance type of g_n, $VT(g_n)$, we make a RVM description $RVM_{VT(g_n),gct(g_n)}$ having the BTU which has the unit size closest to the $gct(g_n)$. We check the following condition to choose whether to lease $RVM_{VT(g_n),gct(g_n)}$ from cloud resource provider or not

$$\frac{C\left(RVM_{VT(g_n),gct(g_n)}\right)}{\sum_{\forall t \in g_n} et(t) \cdot C\left(OVM_{VT(t)}\right)} < 1 \tag{1.5}$$

The denominator of Eq. (1.5) represents the total cost for execution time et the tasks in g_n on OVMs. The numerator of Eq. (1.5) represents the cost of RVM for g_n. If Eq. (1.5) is satisfied, it means that the leasing of RVM is more cost efficient than the leasing of OVMs for g_n. As the equation value in Eq. (1.5) is decreased, the cost efficiency by using RVM is increased (Fig. 1.18).

1.8 Workflow Scheduling Scheme with Division Policy

A SLA constrained scientific application to be executed within user-specified deadline D is defined as workflow scheduling problem with deadline constraint. That is, deciding assigned computing resources-to-be set $R = \{R_1, R_2, \ldots, R_n\}$ and assigned time set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ according to Petrinet model with inter-task dependencies determines performance of Science Gateway.

It is hard to find a schedule for satisfying user-specified tight deadline due to the finite set of resource types in cloud computing. Unlike workflow scheduling with CPU time competition on cluster computing, the on-demand resource provisioning is operated on cloud workflow scheduling. Therefore, the coarse grained resource allocation with finite set of resource types make scheduling schemes to be more sophisticated for better resource utilization. Also, when the deadline is shorter than

Algorithm 2. VMPM resource allocation policy

INPUT : *historical data including* $S = \{\forall t_{i,j}\}$ *and* $\forall VT(t_{i,j})$ *during previous time interval* T'

OUTPUT : N_i during current time interval T

```
For $VT = 1$ to $K$
  For $\forall t \in S$
    $Cls_{VT} = Cls_{VT} \cup\leftarrow t \in S\ if\ VT(t) = VT$
  End for
End for
$n = 0$
For $\forall Cls_{VT}$
  While available $w$ exists do
    $g_n = g_n \cup\leftarrow t' \in Cls_{VT}\ if\ st(t') \geq ft(t'')$,
    $t' = first\ task\ in\ Cls_{VT}, t'' = last\ task$ in $g_n$
  End while
  $n = n + 1$
End for
For $\forall g_n$
  $gct(g_n) = ft(g_n) - st(g_n)$
  $VT = VT(g_n)$
 If $\frac{C(RVM_{VT(g_n),gct(g_n)})}{\sum_{\forall t \in g_n} et(t)\cdot C(OVM_{VT(t)})} < 1$ then
lease $RVM_{VT(g_n),gct(g_n)}$ from cloud resource provider
 End if
End for
```

Fig. 1.18 An algorithm of virtual machine pool management [27]

the earliest possible execution time of a workflow instance or the unexpected processing delays are occurred above certain level, we can't guarantee the deadline in order to the limitation on expressible quantity measure.

A SLA constrained scientific application in Definition 1.1. $S(\Upsilon, \Phi, D)$ is transformed with the Petrinet workflow model $S(W(P, T, A), D)$ where P is place, T is transition and A is arc [28], for better status expression and execution control with mathematical tools.

A task t_i in scientific application, which can be partitioned as they do not have any precedence relations and all sub-tasks are identical, is defined as divisible task [38]. The divisible degree (dd) on divisible task is defined as the maximum number of possible sub-tasks. Then, sub-tasks on divisible task t_i can be illustrated as set $\{t_{i,1}, t_{i,2}, \ldots, t_{i,n}\}$. From the task parallelization, the processing time can be reduced through allocating distributed resources on sub-tasks. Though task division strategy provides time reduction on workflow scheduling, the division is not always carried out as it is not appropriate to reduce required cost and processing time, namely, the goal of workflow scheduling problem in order to the availability of over-provisioning and burdensome management cost. The composition of partial sub-task on task type $\mathrm{tt_k}$ with amount r is defined as sub-task type $tt_{k,\{r\}}$ whose $dd(tt_{k,\{r\}}) = r$. Additional application profiling for all kinds of task bunches is required to measure execution time of each subtask type $tt_{k,\{r\}}$ on different resource type VT_i.

When token forward and backward matrix is already extracted by analysis of workflow topology, the SLA constraints scientific application in Petrinet scheme

Algorithm 3. Division Policy

Input: *workflow topology tp, workflow topology with half division tp', current execution token matrix m, current execution time* $T_C(m)$*, workflow deadline D, budget B, token forward matrix F*

Output: *Decision of division*

```
Copy execution token set m onto SV estimation token set $m_e$ and copy current execution time $T(m)$ onto temporal current execution time $T_t(m_e)$.
For workflow topology {tp, tp'}
    While $m_e$ is not equal to [0 … 0 1] do
        Let P= $\{p_1, p_2, \ldots, p_n\}$ be places which has SV estimation tokens at current stage i and estimation token set at stage i as $m_e^i$
            For p = each element in P
                $sd(p^*) = rl(p) \cdot T_t(D - m_e^i)$
                $p^*.temporal\_vm = VT_{\{j | T_{VM_j}^{type(p^*)} < sd(p^*), min\left(C_{VT_j} \cdot T_{VM_j}^{type(p^*)}\right)\}}$
                Let $Tr = \{t_1, t_2, \ldots, t_k\} =^* p$
                $T_{tc}(m_e^i) = T_{tc}(m_e^{i-1}) + \max_j \left(T_{t_j.temporal\_vm}^{type(t_j)}\right)$
            End for
            $m_e^{i+1} = F \cdot m_e^i$ //token forwarding
    End while
    If $T_{tc}(m_e) - D > 0$ then
        $C_p = \alpha + \beta \cdot SV = \alpha + \beta \cdot (T_{tc}(m_e) - D)$
    Else
        $C_p = 0$
    End if
    $C_l = \sum_i C_{t_i.temporal\_vm} \times T_{t_i.temporal\_vm}^{type(t_i)}$
    $Pf = B - C_l - C_p$
End for
Return $Pf < Pf'$    // If $Pf < Pf'$    , return value is true. Otherwise return value is false
```

Fig. 1.19 An algorithm of task division policy in workflow scheduler [27]

$S(W(P,T,A),D)$ can be controllable by moving the token with multiplication of token status vector and token forward/backward matrix. Whole algorithm is described in Fig. 1.19.

[Phase I] Calculate the load rate r(p) for each placement

Initial marking of token vector is set for first phase as $m = [0..01]$. Then the token moves through backward matrix along the Petrinet topology path reversely while investigating each task's the load rate. The load rate $r(p)$ on a place p is the rate of the following transition's relative load compared to relative load of its critical path.

$$r(p) = \frac{rl(p)}{cpl(p)} \tag{1.6}$$

Also, relative load is defined as average execution time for a task on VM types:

$$rl(p) = \mathrm{avr_j}\left(\tau_{VT_j}^{type(p^*)}\right) = \frac{1}{\mathrm{m}} \cdot \sum_j \tau_{VT_j}^{type(p^*)} \tag{1.7}$$

In addition, Critical path on a task is defined as set of following tasks which is composed of biggest relative load [21]. Then, critical path load is defined as:

$$cpl(p) = f(x) = \begin{cases} \max\limits_{\mathrm{p}^{**}} cpl(p^{**}) + rl(p), & if\, p^{**} exist \\ rl(p), & otherwise \end{cases} \tag{1.8}$$

[Phase II] Allocate the most cost efficient resource with properly assigned sub-deadline

Initial marking of token vector is set for second phase as $m = [10\ldots0]$. Then the token moves through forward matrix along the Petrinet topology path while allocating cheapest VM which can guarantee estimated sub-deadline. To achieve successful scheduling with guarantee of the entire deadline, properly assigned sub-deadline for each task should be observed. Therefore, we allocate sub-deadline (sd) rationally based on remaining time and load rate when T(m) is execution timestamp of with token status m.

$$sd(p^{*}) = rl(p) \cdot (D - T(m)) \tag{1.9}$$

When the leasing cost per unit time for arbitrary VM type VT_i is illustrated as C_{VT_i} and there are known estimated time for task execution time on each VM type $\mathrm{T}_{\mathrm{VM_i}}^{type(p^{*})}$, the most efficient resource with guaranteeing sub-deadline for j-th task can be determined through leasing VM with type of $\mathrm{VT}_{\left\{i|\mathrm{T}_{\mathrm{VM_i}}^{type(p^{*})} < sd(p^{*}), \min\left(C_{VT_i} \cdot \mathrm{T}_{\mathrm{VM_i}}^{type(p^{*})}\right)\right\}}$.

In case of no available resource types to guarantee sub-deadline, it may cause deadline violation for entire workflow.

[Phase III] Task division with profit calculation

To compare evaluation of solutions on constrained problem only in objective domain, the service level violation penalty on SLA constraint can be considered. Profit cost model can be defined to maximize profit while proceeding task division:

$$Pf = B - C_l - C_p \tag{1.10}$$

In the formula above, Pf indicates Profit. B is budget which is supplied by user. C_l is total cost for leasing VM from cloud service providers, $C_l = \sum_i R_i \cdot \tau_i$. Penalty cost C_p, which is caused by SLA violation SV, is represented as follows:

$$C_p = \begin{cases} \alpha + \beta \cdot SV, & if\, SV > 0 \\ 0, & otherwise \end{cases} \tag{1.11}$$

$$SV = \begin{cases} ECT - D, & \mathit{if}\ ECT - D > 0 \\ 0, & \mathit{otherwise} \end{cases} \tag{1.12}$$

In Eqs. (1.11) and (1.12), variable SV indicates the degree of SLA violation. There are many models of violation penalty cost, but in this book we use linear violation penalty model in Eq. (1.11) [39]. As shown in Eq. (1.12), SV can be described as subtraction deadline D from estimated completion time ECT.

By estimating SLA violation SV over non-division case and SV' over division case in a deterministic way, the profit for making decision of division can be considered. The penalty cost can be calculated by setting SV Estimation Token $\mathrm{m_e}$ which is clone of current token status m. Initially, the location of $\mathrm{m_e}$ is replicated from current execution token m. Also, each token save temporal execution timestamp $T_t(m_e)$ which is replicated from current execution time $\mathrm{T}(m)$. By forwarding SV estimation token with allocation of temporal VM over the estimated-sub-deadline, we can cumulate the $\mathrm{T_t}(m_e)$ for each transition with the scheme described in phase II until token reach the final place. Then the estimated execution time ECT from timestamp $\mathrm{T_t}(m_e)$ can be obtained in heuristic way. With the comparison of profit between the non-division case and division case (half division), cost efficient decision can be available.

If $Pf < Pf'$, apply the half division and return to phase II. If division case not yet guarantees the deadline, division can be occurred recursively until task is no further divisible (divisible degree equals 1). Otherwise, allocate biggest VM to transition and return to phase II for forwarding token to next place P**.

1.9 Test Environments for Performance Evaluation on Resource Management Schemes of the Science Gateway

To evaluate the performance of the science gateway, the test environment with heterogeneous cloud platforms was built as Fig. 1.20 and specification on test environment is shown in Table 1.6. The two cloud platforms were coordinated using 5 computing nodes for the OpenStack and 4 computing nodes for the CloudStack respectively [40, 41]. These platforms provide the VM Lifecycle management (i.e. create, terminate, pause, reboot, snapshot) through the network service, volume service, scheduling service, image service and virtualization. To implement the heterogeneity in the multiple cloud services (e.g. Amazon EC2, GoGrid, Windows Azure), different hardware and software for each platform is considered. The OpenStack platform is coordinated with Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz, Core 16, MEM 16G and the software with KVM hypervisor, Ubuntu 14.04 OS. On the other hands, the CloudStack platform is coordinated with Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz, Core 8 with hyper threading, MEM 16G, HDD 1T and the software with XEN hypervisor, CentOS 6.0. The available

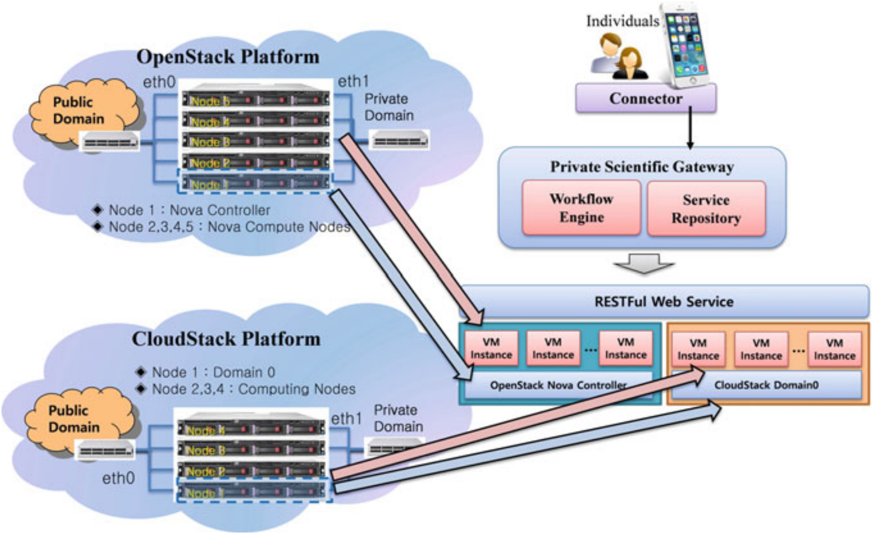


Fig. 1.20 Experimental testbed of the science gateway [27]

Table 1.6 Specific configurations on testbed environment [27]

		OpenStack platform	CloudStack platform
Hypervisor		KVM	XEN
H/W spec		Intel Xeon E5620 2.40 GHz, Core 16, MEM 16G, HDD 1T, 5 Node	Intel Core i7-3770 CPU 3.40 GHz, Core 8, MEM 16G, HDD 1T, 4 Node
S/W spec		OS: Ubuntu 14.04	OS: CentOS 6.0
VM types	small	Spec: 1 VCPU, 2 GB MEM, 80 GB Disk, Unit Time Cost: 2 RC per second	
	medium	Spec: 2 VCPU, 4 GB MEM, 80 GB Disk, Unit Time Cost: 4 RC per second	
	large	Spec: 4 VCPU, 8 GB MEM, 80 GB Disk, Unit Time Cost: 8 RC per second	
	c4.small	Spec: 4 VCPU, 1 GB MEM, 80 GB Disk, Unit Time Cost: 4 RC per second	
	m8.small	Spec: 1 VCPU, 8 GB MEM, 80 GB Disk, Unit Time Cost: 4 RC per second	

VM types in these cloud platforms are small type (1 CPU, 2 GB MEM, 80 GB Disk), medium type (2 CPU, 2 GB MEM, 80 GB Disk), large type (4 CPU, 4 GB MEM, 80 GB Disk), MEM intensive type (1 CPU, 4 GB MEM, 80 GB Disk), CPU intensive type (4 CPU, 1 GB MEM, 80 GB Disk).

To evaluate the cost efficiency, relative cost that does not have monetary meaning in reality but the cost has theoretical meaning for comparison between algorithms or models is defined. The relative cost is defined as a unit for the numeric value of cost. To determine the metric on relative cost, we consider the unit time cost. The unit time cost is amount of payment on specific time period for imposing price on service that is agreed on the contract. Then, the definition of relative cost in the *i*th contract is described as multiplying unit time cost on the *i*th contract c_u^i by unit time consumption on the *i*th contract t_u^i and addition of constant value *a*, where i is the sequence number of resource contract for the execution of specific workflow.

$$\mathrm{RC}^\mathrm{i} = \mathrm{c}_\mathrm{u}^\mathrm{i} \cdot t_u^i + a \tag{1.13}$$

The contracts can be made from each resource contract on the workflow scheduling, also from the imposing penalty on service level violation and even from the long-term VM reservation. Unlike the billing contract with hourly policy on real cloud service domain, we assign unit time as a second for the minute examination on the schemes. When the specification of resource can be simplified as a tuple $[r_c^i, r_m^i]$ of *i*th resource contract (where r_c^i is the number of CPU cores and r_m^i is the numeric value of RAM size in GB) and the weight vector $\vec{w} = [w_c, w_m]$ to apply the effectiveness of each element on tuple is presented, the unit time cost on resource contract, c_{ur}^i is calculated as follows.

$$c_{ur}^i = w_c \cdot r_c^i + w_m \cdot r_m^i \tag{1.14}$$

In case the weight vector is assigned as [0, 0.5], the unit costs for each of VM instance types (i.e. small, medium, large, c4.small and m8.small) are 2, 4, 8, 4 and 4 per second respectively. Then, relative cost on resource contract can be represented as Eq. (1.15), where τ_r^i is unit time consumption on *i*th resource contract.

$$c_r^i = c_{ur}^i \cdot \tau_r^i \tag{1.15}$$

In addition, the relative cost on *j*th service level violation is defined as following Eq. (1.16) by using the penalty cost model in Eq. (1.12) where j is the sequence number of the workflow, c_{sv}^j is unit time cost on service level violation, τ_{sv}^j is degree of service level violation and α is constant relative cost imposed per violation.

$$c_{sv}^j = \begin{cases} c_{usv}^j \cdot \left(\tau_s^j - \tau_{sl}^j\right) + \alpha, \tau_s^j - \tau_{sl}^j > 0 \\ 0, \qquad\qquad\qquad\qquad otherwise \end{cases} \tag{1.16}$$

In this experiment, Next Generation Sequencing (NGS) with Burrows-Wheeler Aligner (BWA) was orchestrated as a scientific workflow. The NGS is used for the determination of the order on the nucleotide bases in DNA molecules. In addition, it is also used for the analysis of the resulting sequences to understand biological phenomena and relation between genotype and phenotype. Especially, BWA is pipelined set of tasks for analyzing genome by using burrows-wheeler transform compression techniques. Similar to typical scientific application, BWA is also computing/data intensive, time consuming and divisible in some degree [42]. From the application profiling on BWA applications, data collected on each service show performance diversity over the distribute cloud resources with the heterogeneity on both hardware and software. Also, the application profiling on divisible tasks with the task parallelization are conducted for implementation of workflow scheduling scheme with division policy.

1.10 Performance Evaluation on Resource Management Schemes of Science Gateway

To evaluate the performance of workflow scheduling scheme with the division policy over the cost effectiveness on RVM management for scientific application in heterogeneous cloud environment, 5 different experiments was performed as described in Table 1.7. The experiment 1, 2, 3 are designed to figure out the properties of workflow scheduling scheme with the division policy over the variation of multiple parameters which effect on system performance. In addition, the experiment 4, 5 are designed to figure out the effectiveness of cost adaptive resource management scheme with long-term VM reservation. In the experiments, multiple users make contracts of workflow execution with reservation of SLA (deadline) and determination of parameters on the penalty cost function. To identify the conditions on experiments, each deadline and penalty cost function are configured as equivalent distribution over the entire contracts.

Table 1.7 A parameter sets and performance metrics for the experiments [27]

Parameters	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5
Average interarrival (Exp(1/λ), (s))	[200, 300, …, 900]	[200, 300, …, 900]	[200, 300, …, 900]	100	[100, 150, …, 500]
Service level (deadline, (s))	[500, 700, 900, 1100]	800	800	700	700
Penalty function	β = 5, α = 0	β = [1, 3, 5, 7], α = 0	β = 5, α = 0	β = 5, α = 0	β = 5, α = 0
Artificial delay factor γ	1.0	1.0	[1.0, 1.2, 1.4,1.6]	1.0	1.0
Number of initial RVM	0	0	0	[4, 6, …, 22]	10

Performance metric	
Resource contract cost (relative cost)	$c_r = \sum_i c_{ur}^i \cdot \tau_r^i + \mathrm{n}_{\mathrm{rvm}} \cdot c_{urvm} \cdot \tau_{total}$
Service level violation (penalty) cost (relative cost)	$c_{sv} = \sum_{j\mid\tau_s^j-\tau_{sl}^j > 0} \left(c_{u\mathrm{sv}}^j \cdot \left(\tau_s^j - \tau_{sl}^j\right) + \alpha\right) = \sum_{j\mid t_{sv}^j > 0} \left(\beta \cdot \tau_{sv}^j + \alpha\right)$
Average service level violation (s)	$avr(\tau_{sv}^j)$
Total cost (relative cost)	$c_t = c_r + c_{sv}$
Cost improvement rate	$cost\ improvement\ rate = \frac{Relative\ Cost\ with\ RVM}{Relative\ Cost\ without\ RVM}$

Through the experiments, Petrinet based dynamic workflow scheduling scheme with division policy is evaluated against Petrinet based dynamic workflow scheduling scheme [43] while overcoming the drawbacks on previous solution, especially the ability to react on the expectable service level violation and the unexpected failures.

In experiment 1, with different average interarrival time of contract on workflow execution in exponential distribution from 200 to 900 s, relative cost on service level violation is measured according to the difference SLA requirement on both ‘without division policy’ in Fig. 1.21a and ‘with division policy’ in Fig. 1.21b. Because there are no dependencies between the workflow scheduling by avoiding competition on resource due to the provision of abundance resource from the multiple cloud services over the worldwide providers, it can obviously be figured out that the decline of relative cost over the longer interarrival time according to the little workload when experiment times are same. When we compare Fig. 1.21a, b, scheduling with division policy shows lower cost demand for same workload and we can figure out about 85% decrement of service level violation cost in average by overcoming the tight deadline through the enhancement of performance with task parallelization, although the result might be workload dependent.

In experiment 2, with different average interarrival time of contract on workflow execution in exponential distribution from 200 to 900 s, relative cost on service level violation was measured according to the difference parameters on the penalty cost function onto the both ‘without division policy’ in Fig. 1.22a and ‘with division policy’ in Fig. 1.22b. When user impose excessive penalty cost on service level violation, the division policy might try to avoid the violation from intensive rate of division although there might be more leasing cost on leasing resource. On the other hand, when insignificant penalty cost function are imposed, the division policy might not divide the task from the comparison of penalty cost and leasing cost in profit model described in Eq. (1.10). The beta value indicates the imposing relative cost per second for service level violation. Unlike increment of the violation cost in proportion to beta in Fig. 1.22a, the division policy in Fig. 1.22b select best

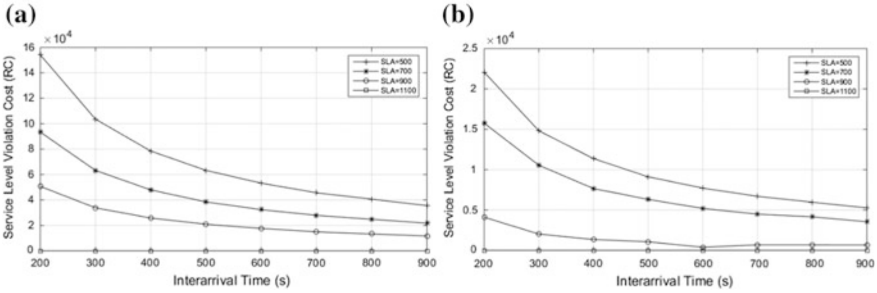


Fig. 1.21 Experiment 1—relative cost on service level violation w.r.t. different service level requirements as 500, 700, 900, 1100 s (i.e. deadlines are set as 500, 700, 900, 1100 s by user respectively): **a** without division policy **b** with division policy [27]

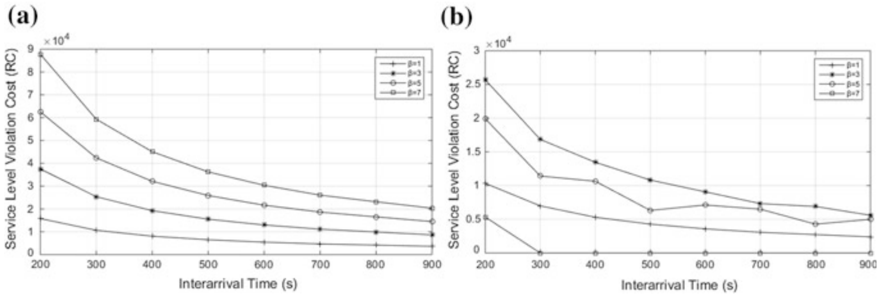


Fig. 1.22 Experiment 2—relative cost on service level violation w.r.t. different parameters on the penalty cost function as β = 1, 3, 5, 7 (refer to Eq. (1.12)) **a** without division policy **b** with Division Policy [27]

solution of division with the profit cost model. In addition, experiment shows extreme case of no service level violation when beta is 7 due to radical division derived from the excessive penalty. Also result shows about 50% improvement of cost efficiency on service level violation in average compared to workflow scheduling without division.

In experiment 3, with different average interarrival time of contract on workflow execution in exponential distribution from 200 to 900 s, average service level violation is measured according to the difference artificial delay factor γ on each task for both 'without division policy' in Fig. 1.23a and 'with division policy' in Fig. 1.23b. To figure out the influence of unexpected failures or delays in distributed cloud computing, artificial delay is simulated with imposing arbitrary delay on workflow by multiplying artificial delay factor γ on execution time of each task in the workflow instance. Then, it can be figured out that the case without division can't handle the additional burden from delay and show result of excessive service level violation. On the other hand, the scheme shows its capability to handle unexpected failures in some degree.

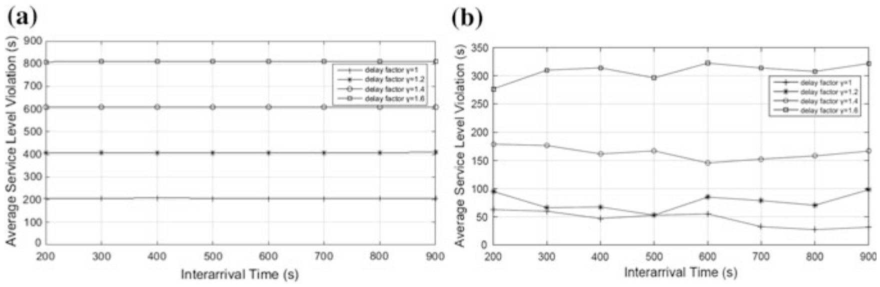


Fig. 1.23 Experiment 3—average service level violation w.r.t. added artificial delay factor γ as 1, 1.2, 1.4, 1.6 (i.e. workflow with artificial delay is obtained by multiplying γ on execution time of each task in workflow instance) **a** without division policy **b** with division policy [27]

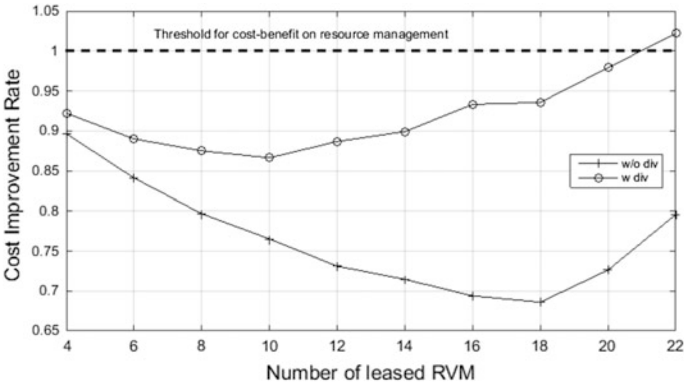


Fig. 1.24 Experiment 4—cost improvement rate with RVM management according to the number of leased RVM [27]

In experiment 4 (Fig. 1.24), with different number of leased RVM in resource pool from 4 to 22, cost improvement rate (refer to Table 1.7) is measured according to the different scheduling policies when unit cost of RVM is half on same type of On-demand VM. From the cost efficiency on RVM with long term leasing contract, we can operate cost adaptive resource management system with the RVM management. However, the reason why RVM can't be operated with On-demand manner, it can waste utilization of resource when there is idle, waiting VM exist. The experimental result shows the case of over-provisioning on RVM when initial number of RVM is 22 and policy without division, while exceeding the threshold of cost-benefit ratio, 1.0. In addition, the result also show that division case gains less efficiency then the case without division. From the detail analysis on raw data, we can figure out that burst request from the division affect negative influence on resource management although there are same workloads from either case. Also we can figure out from the result that there are optimal solutions which are in convex shaped function and can be variable as workload changed.

In experiment 5 (Fig. 1.25), with different average interarrival time of contract on workflow execution in exponential distribution from 100 to 500 s, cost on resource contract is measured according to the different policies, to figure out the effect on RVM management over various policies. Obviously, we can figure out the different influence of cost adaptive resource management on division policies over the various interarrival times. Generally, policy without division shows better synergy then the other, and also there are different threshold of cost-benefit similar to experiment 4. However, when we consider the total cost calculated by addition of resource contract cost and service level violation cost, division policy always shows better cost efficiency then the other.

Through the experiment, the cost efficiency and deadline guaranteeing of workflow scheduling with division policy was figured out. Besides, from experiment 2, we can assure that if there are more excessive penalties on service level

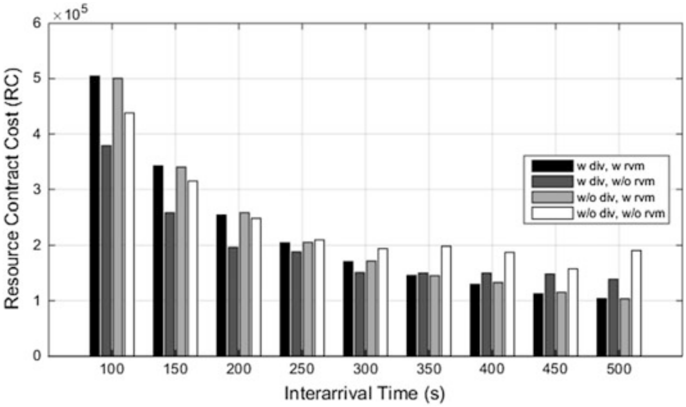


Fig. 1.25 Experiment 5—relative cost on resource contract with different scheduling policies (i.e. with/without division policy, with/without RVM management) [27]

violation, the scheme more extremely divides the tasks for guaranteeing SLA reasonably. In addition, we can figure out the relationship of workload and cost adaptive RVM management in VMPM with convex optimal. In the experiments, division policy always shows about 20% better efficiency in average on the experimental environment.

References

1. J. Yu, R. Buyya, C.C.K. Tham, Cost-based scheduling of scientific workflow applications on utility grids. e-Science Grid Comput. **2005**, 1–8 (2005)
2. Amazon EC2 Pricing. [Online]. Available: http://aws.amazon.com/ec2/pricing/
3. Y. Han, *A Study on Adaptive Resource Management System based on Active Workflow Control Scheme in Distributed Computing* (KAIST, Daejeon, South Korea, 2011)
4. C. Kenyon, G. Cheliotis, Architecture Requirements for Commercializing Grid Resources. *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing* (2002)
5. Document for web services agreement specification. [Online]. Available: http://www.ogf.org
6. D.-S. Nam, C.-H. Youn, B.-H. Lee, G. Clifford, J. Healey, QoS-constrained resource allocation for a grid-based multiple source electrocardiogram application. Lect Notes Comput Sci **3043**, 352–359 (2004)
7. C.-H. Youn, B. Kim, D. S. Nam, B.-H. Lee, E.B. Shim, G. Clifford, J. Healey, Resource reconfiguration scheme based on temporal quorum status estimation in computational grids. *International Conference on Information Networking*, 699–707 (2004)
8. S. Deng, *A Study on Policy Adjuster Integrated Grid Workflow Management System* (Information and Communication University, Daejeon, Republic of Korea, 2008)
9. H. AlHakami, H. Aldabbas, T. Alwada'n, Comparison between cloud and grid computing: review paper, Int. J. Cloud Comput. Serv. Archit., **2**(4), 1–21, 2012
10. Top 10 Strategic technology trends for 2014, (2014) [Online]. Available: http://www.gartner.com

11. F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, D. Leaf, NIST cloud computing reference architecture recommendations of the National Institute of Standards. Nist. Spec. Publ. **292**(9), 35 (2011)
12. H. Kim, Y. Ha, Y. Kim, K. Joo, C.-H. Youn, A VM Reservation-Based Cloud Service Broker and Its Performance Evaluation, in *Cloud Computing: 5th International Conference, CloudComp 2014, Guilin, China, 19-21 Oct 2014, Revised Selected Papers*, ed. by V.C.M. Leung, R.X. Lai, M. Chen, J. Wan (Springer International Publishing, Cham, 2015), pp. 43–52
13. W.-J. Kim, An Integrated Broker System for Policy-based Application Service Management in Mobile Cloud (KAIST, Daejeon, 2014)
14. M. Chen, S. Mao, Y. Zhang, V. C. M. Leung, *Big Data: Related Technologies, Challenges And Future Prospects*. Springer, Berlin (2014)
15. M. Chen, S. Mao, Y. Liu, Big data: a survey. Mob. Netw. Appl. **19**(2), 171–209 (2014)
16. Y. Ren, A Cloud Collaboration System with Active Application Control Scheme and Its Experimental Performance Analysis (Korea Advanced Institute of Science and Technology, Daejeon, 2012)
17. B. Kim, A Study on Cost Adaptive Cloud Resource Broker System for Bioworkflow Computing (Korea Advanced Institute of Science and Technology, Daejeon, 2013)
18. M. Mao, M. Humphrey, Scaling and Scheduling to Maximize Application Performance Within Budget Constraints in Cloud Workflows. In *Proceedings of IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 67–78 (2013)
19. M. Mao, M. Humphrey, Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *2011 International Conference for High Performance Computing Networking Storage and Analysis SC*, pp. 1–12 (2011)
20. Science Gateway. [Online]. Available: http://www.sciencegateway.org
21. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, Basic local alignment search tool. J. Mol. Biol. **215**(3), 403–410 (1990)
22. K. Byungsang, A study on cost adaptive cloud resource broker system for bioworkflow computing (Korea Advanced Institute of Science and Technology, Daejeon, 2013)
23. BLAST. [Online]. Available: http://blast.ncbi.nlm.nih.gov
24. BWA. [Online]. Available: http://bio-bwa.sourceforge.net
25. A. Matsunaga, M. Tsugawa, J. Fortes, CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications, In *Proceedings of 4th IEEE International Conference on eScience, eScience 2008*, pp. 222–229 (2008)
26. C. Vecchiola, S. Pandey, R. Buyya, High-Performance Cloud Computing: A View of Scientific Applications. In *I-SPAN 2009—The 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pp. 4–16 (2009)
27. S.-H. Kim, D.-K. Kang, W.-J. Kim, M. Chen, C.-H. Youn, A science gateway cloud with cost-adaptive VM management for computational science and applications. IEEE Syst. J. **11** (1), 1–13 (2016)
28. Z. Su, B. Ning, H. Fang, H. Hong, R. Perkins, W. Tong, L. Shi, Next-generation sequencing and its applications in molecular diagnostics. Expert Rev. Mol. Diagn. **11**(3), 333–343 (2011)
29. 종필최, “차세대 염기서열 정렬 도구 소개,” 질병관리본부 유전체센터 바이오과학정보과. [Online]. Available: http://www.cdc.go.kr/CDC/cms/content/30/12630_view.html
30. H. Li, N. Homer, A survey of sequence alignment algorithms for next-generation sequencing. Brief. Bioinform. **11**(5), 473–483 (2010)
31. SAMtools. [Online]. Available: http://sourceforge.net/mailarchive/forum.php?thread_name_2F0E69A8-A2DD-4D6E-9EDE-2A9C0506DA0F%2540sanger.ac.uk&forum_name=samtools-devel
32. D. Sulakhe, M. D’Souza, M. Syed, A. Rodriguez, Y. Zhang, E.M. Glass, M.F. Romine, N. Maltsev, GNARE—a grid-based server for the analysis of user submitted genomes. NAR 00335 (2007)

33. N. Maltsev, E. Glass, D. Sulakhe, A. Rodriguez, M.H. Syed, T. Bompada, Y. Zhang, M. D'Souza, PUMA2–grid-based high-throughput analysis of genomes and metabolic pathways. Nucleic Acids Res. **34**, D369–D372 (2006) (Database issue)
34. J. Zhang, J. Yao, S. Chen, D. Levy, Facilitating biodefense research with mobile-cloud computing. Int. J. Syst. Serv.-Oriented Eng., **2**(3), 18–31 (2011)
35. B. Kim, C.-H. Youn, An Adaptive Resource Provisioning Scheme for Distributed Bio-workflow Broker with Stream-Based NGS in Cloud. *In the 6th International Conference on Ubiquitous Information Technologies and Applications* (2011)
36. S. Chaisiri, S. Member, B. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing. Computer (Long. Beach. Calif)., **5**(2), 1–32 (2012)
37. B. Jennings, R. Stadler, Resource Management in clouds: survey and research challenges. J. Netw. Syst. Manag. **23**(3), 567–619 (2014)
38. Z. Xiao, Z. Ming, A method of workflow scheduling based on colored Petri nets. Data Knowl. Eng. **70**(2), 230–247 (2011)
39. N.J. Malawki M., Juve G., Deelman E, Cost and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. IEEE Int. Conf. High Perform. Comput. Networking, Storage Anal. **48**, 1–11 (2012)
40. OpenStack. [Online]. Available: http://www.openstack.org/
41. Cloudstack. [Online]. Available: http://cloudstack.apache.org
42. Z. Su, B. Ning, H. Fang, H. Hong, R. Perkins, W. Tong, L. Shi, Next-generation sequencing and its applications in molecular diagnostics. Expert Rev. Mol. Diagn. **11**(3), 333–343 (2011)
43. D.-S. Kim, Adaptive Workflow Scheduling Scheme Based on the Colored Petri-net Model in Cloud (KAIST, Daejeon, 2014)

Chapter 2
VM Placement via Resource Brokers in a Cloud Datacenter

2.1 Introduction

Resource management in cloud datacenters is one of the most important issues for cloud service providers because it directly affects their profit. Energy and performance guarantee are two major concern of it. In energy aspect, the total estimated energy bill of datacenters is $11.5 billion and their energy bills double every five years [1, 2]. Also, in performance guarantee aspect, many researches insist that performance metrics such as throughput and response time should be considered as well as availability in IaaS SLA [3, 4]. If the IaaS SLA with the performance metrics are applied in public CSPs, performance management should be much more delicate to avoid the SLA penalty cost. Especially in VM placement in the cloud, an application quality of service (QoS)-based approach to allocate workload fairly in physical machines (PMs) and a power-based approach to consolidate VMs maximally are two basic goals respectively [5]. To reduce energy consumption in a cloud datacenter, dynamic right sizing (DRS) is a promising technology to dynamically adjust the number of active servers (i.e., servers whose power is switched on) in proportion to the measured user demands [6]. In DRS, energy saving can be achieved by enabling idle compute nodes that do not have any running VM instances to go into low-power mode (i.e., sleep or shut down). In order to maximize the energy efficiency via DRS, one of the primary adaptive resource management strategies is VM consolidation in which running VM instances can be dynamically integrated into the minimal number of compute nodes based on their resource utilization collected by a hypervisor monitoring module [7]. However, it is difficult to efficiently manage cloud resources because cloud users often have heterogeneous resource demands underlying multiple service applications which experience highly variable workloads. Therefore, inconsiderate VM consolidation might lead to undesirable performance degradation due to workload overloading. Although A virtualization technology allows CSPs to get many benefits such as getting high PM resource utilization via server consolidation and

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_2

elasticity in their resource usage. In virtualized environments, to guarantee performance in VMs sharing a PM, a hypervisor should provide isolation between VMs. However, while security isolation, fault isolation, and environment isolation are well guaranteed, the current virtualization technology does not provide effective performance isolation. Therefore, the VMs make an effect with each other and it causes performance degradation in the VMs. The phenomenon is called as performance interference [8].

Therefore, addressing the conflict between VM consolidation and dispersion is essential for effective cloud resource management. To achieve it, Patal and Shah [9] argued the need of cost modeling of a datacenter, and constructed the cost model. In the model, the total cost in a cloud datacenter includes the space, the power and the cooling recurring, and other cost. The other cost consists of maintenance and amortization of power and cooling system, personnel, software licenses, compute equipment depreciation, and so on.

In addition, we consider IaaS service level agreement (SLA) cost to the model. Although almost all cloud service providers only consider availability of VMs as IaaS SLA today, necessity of considering performance in IaaS SLA is raising as in [3, 4]. To achieve it, we define the SLA penalty cost (*PC*) of compute node *j*, $PC_j(t)$ as depicted in Eq. (2.1) where *UPC* is the unit *PC*, $v_j(t)$ is resource usage of VMs in timeslot *t*, and $\overline{d}_j(t)$ is the average performance degradation in timeslot *t*.

$$PC_j(t) = UPC \cdot v_j(t) \cdot \overline{d}_j(t). \tag{2.1}$$

Finally, we describe the partially total cost (*PTC*) which is composed of costs only depending on VM placement in the total cost as depicted in Eq. (2.5). We note that the *OC* is an abbreviation of the PM operating cost including the power and the cooling recurring cost. In Eq. (2.2), *J* is the set of PMs in a cloud datacenter, *UOC* is the unit *OC*, $V_j(t)$ is a vector of $v_j(t)$, and $p_j(V_j(t))$ is power consumption of node *j* when resource usage in timeslot *t* is $V_j(t)$.

$$\begin{aligned} PTC(t) &= \sum_{j\in J} OC_j(t) + PC_j(t) \\ &= \sum_{j\in J} UOC \cdot p_j\big(V_j(t)\big) + UPC \cdot v_j(t) \cdot d_j(t). \end{aligned} \tag{2.2}$$

Based on the cost model, we formulate an optimization problem as depicted in Eqs. (2.3) and (2.4). The objective is to minimize the PM operating cost while keeping performance degradation less than the threshold.

$$\text{minimize} \quad \limsup_{t\to\infty} \frac{1}{t} \sum_{t} \sum_{j\in J} OC_j(t) \tag{2.3}$$

$$\text{subject to} \quad \limsup_{t\to\infty} \frac{1}{t} \sum_{t} \sum_{j\in J} PC_j(t) < \delta. \tag{2.4}$$

In this chapter, we handle VM placement schemes to solve the optimization problem. First, we present schemes for computing-aware initial VM placement. The computing-aware initial VM placement is operated to select the appropriate compute nodes to build VMs for executing applications. The placement algorithm is based on execution time prediction of target applications, and the prediction is achieved using methods for application and computing resource profiling and similarity analysis (Sect. 2.2). Second, we present schemes for VM reallocation based on resource utilization-aware two interactive actions: VM consolidation and dispersion. In the schemes, resource utilization of each compute node is carefully predicted by self-adjusting workload prediction. Based on the prediction, the actions are operated to balance VM consolidation and dispersion (Sect. 2.3).

2.2 Computing-Aware Initial VM Placement

2.2.1 Overview

In this chapter, we handle VM placement schemes to solve the optimization problem. First, we present a computing-aware initial VM placement algorithm. In the algorithm, each computing resource is ranked in order of execution performance of each application using profiled information, and the ranking is used for the selection of the appropriate compute nodes to build VMs for executing the applications. We note that that some part of this section is composed based on [10].

2.2.2 Computing-Aware Initial VM Placement Algorithm

Computing-aware initial VM placement is operated in four steps: (1) application profiling, (2) computing resource profiling, (3) similarity analysis, (4) initial VM placement. Given applications and computing resources, application profiling extracts each application's characteristics, and computing resource profiling records execution results of numerous benchmarks. Measuring similarity between each application and the numerous benchmarks, we can estimate execution performance of each application in each computing resource. Finally, initial VM placement is operated based on the estimated performance information.

Application profiling. Application profiling is achieved by extract micro-architecture independent characteristics (MICs) of each application. The MICs provide hardware-independent information of applications and distinguishes each application across various microarchitectures [11]. In Ref. [11], the authors seven categories of MICs as follows.

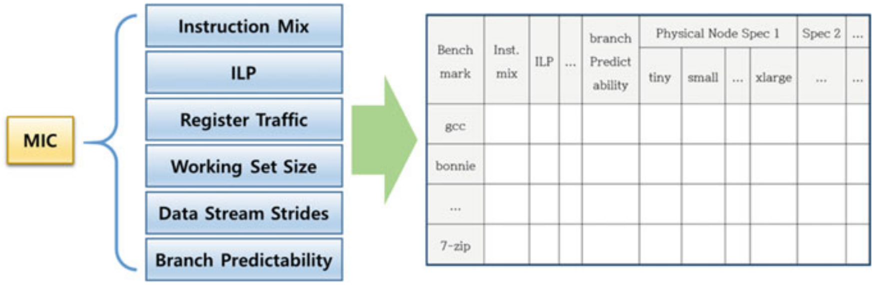


Fig. 2.1 MICs extraction

itypes: represent the percentage of load instruction, store instruction, branch arithmetic operation instruction.

Reg: characterize the register. Measure the utilization of all operation to write to or read from register and register dependency distance.

PPM: measure the accuracy of branch prediction on theoretical prediction-by-partial-matching (PPM).

ILP: represent the rate of the operation which is applicable to simultaneous processing inherent in application.

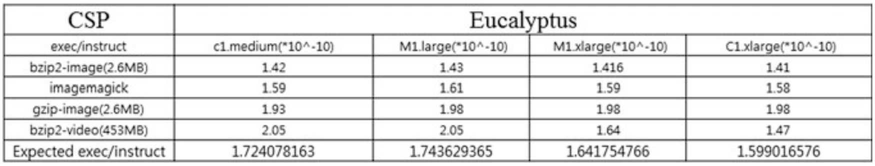
Memreusedist: represent the cache behavior of application and measure the re-usage distribution on memory.

memfootprint: measure the instruction and working set size of data stream.

Stride: measure the difference of memory access interval when successive memory access occurs (Fig. 2.1).

Computing resource profiling. Computing resource profiling is achieved by numerous executions of benchmarks for each compute node. The execution results are normalized by dividing the execution times into the number of instructions (Fig. 2.2).

Similarity analysis. Figure 2.3 shows a procedure of program similarity-based execution time prediction. In the figure, execution time of each application is predicted by measuring similarity between the application and the benchmark sets used for computing resource profiling. Similarity S_{t,b_i} is defined as the reciprocal of Euclidean distance between MIC vectors of the target application t and benchmark b_i. The MIC vectors are defined as $MIC_a = \left(C_1^a, C_2^a, \ldots, C_n^a\right)$ for application a where



CSP	Eucalyptus			
exec/instruct	c1.medium(*10^-10)	M1.large(*10^-10)	M1.xlarge(*10^-10)	C1.xlarge(*10^-10)
bzip2-image(2.6MB)	1.42	1.43	1.416	1.41
imagemagick	1.59	1.61	1.59	1.58
gzip-image(2.6MB)	1.93	1.98	1.98	1.98
bzip2-video(453MB)	2.05	2.05	1.64	1.47
Expected exec/instruct	1.724078163	1.743629365	1.641754766	1.599016576

Fig. 2.2 Computing resource profiling example of memcoder [12] in Eucalyptus [13]

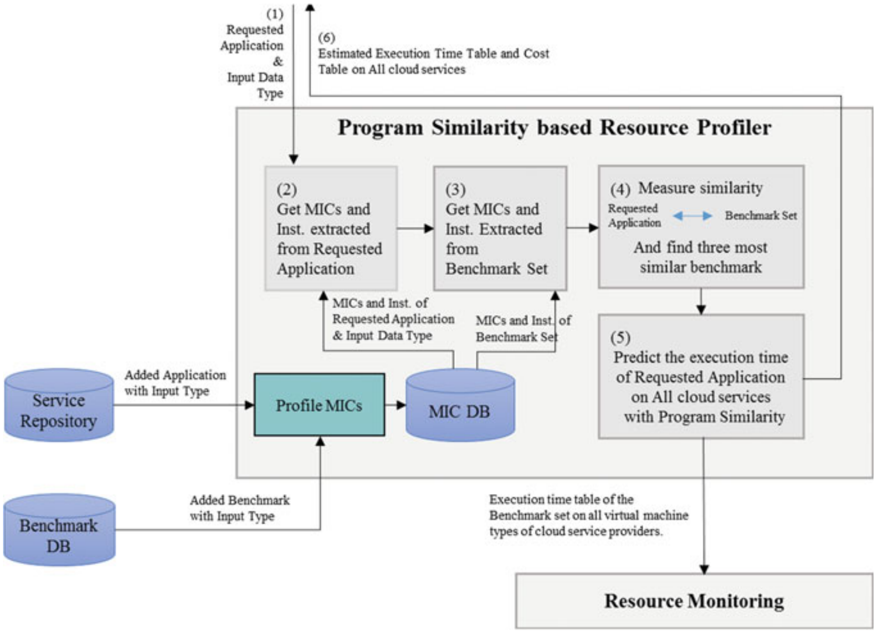

(1)
Requested Application & Input Data Type
(6)
Estimated Execution Time Table and Cost Table on All cloud services
Program Similarity based Resource Profiler
(2)
Get MICs and Inst. extracted from Requested Application
(3)
Get MICs and Inst. Extracted from Benchmark Set
(4) Measure similarity
Requested Application
Benchmark Set
And find three most similar benchmark
MICs and Inst. of Requested Application & Input Data Type
MICs and Inst. of Benchmark Set
(5)
Predict the execution time of Requested Application on All cloud services with Program Similarity
Service Repository
Added Application with Input Type
Profile MICs
MIC DB
Benchmark DB
Added Benchmark with Input Type
Execution time table of the Benchmark set on all virtual machine types of cloud service providers.
Resource Monitoring

Fig. 2.3 A procedure of program similarity-based execution time prediction

each element is the categories of MICs. After measuring the similarities, the resource profiler selects the three benchmarks $\left(b_x, b_y, \ldots, b_z\right)$ which have the highest similarity values, and predicts the execution time of the target application t in computing resource r_j using the measured similarity. Equation (2.5) shows the predicted execution time of the target application t in computing resource r_i where T_{a,r_j} is the execution time of application a in computing resource r_i, $inst_a$ is the number of instructions of application a. The formula is based on [14]

$$\hat{T}_{t,r_i} = \frac{inst_t}{S_{t,b_x} + S_{t,b_y} + S_{t,b_z}} \cdot \left(\frac{S_{t,b_x} \cdot T_{b_x}}{inst_{b_x}} + \frac{S_{t,b_y} \cdot T_{b_y}}{inst_{b_y}} + \frac{S_{t,b_z} \cdot T_{b_z}}{inst_{b_z}} \right) \tag{2.5}$$

Computing-aware VM placement. Algorithm 1 shows computing-aware placement algorithm. In the algorithm, available compute nodes are sorted based on the rank of estimation time prediction results of the target application t in computing resource r_j in ascending order. Then, the algorithm selects an available compute node who has the highest rank.

Algorithm 1. Computing-aware VM placement [10]

```
Input  $t$, $r_i$ ($t$: the target application, $r_i$: computing resource type)
get the sorted compute node list based on execute time prediction results of the target application $t$ in each computing resource $r_j$
for each compute node $p_i$
    if compute node $p_i$ is available for computing resource type $r_i$
        $createVM=createNewVM(p_i,r_i)$
        Return $createVM$
    end if
end for
```

2.3 VM Reallocation Based on Resource Utilization-Aware VM Consolidation and Dispersion

2.3.1 Overview

In this section, we propose a Two Phase based Adaptive Resource Management (TP-ARM) scheme to address the above challenges for cloud datacenters. The energy consumption model based on TP-ARM was formulated with a performance cost (reputation loss) caused by an increased delay from downsizing active servers, by an energy costfrom keeping particular servers active, and by a cost incurred from switching off servers on. Subsequently, we designed an automated cloud resource management system called the Adaptive Cloud Resource Broker (ACRB) system with the TP-ARM scheme. Moreover, we introduced our novel prediction method called Self-Adjusting Workload Prediction (SAWP) to increase the prediction accuracy of users' future demands even under unstatic and irregular workload patterns. The proposed SAWP method adaptively scales the history window size up or down according to the extracted workload's autocorrelations and sample entropies which measure the periodicity and burstiness of the workloads [15]. To investigate the performance characteristics of the proposed approaches, we conducted various experiments to evaluate energy consumption, resource utilization and completion time delay by live migration and DRS execution on a real testbed based on Openstack which is a well-known cloud platform using KVM hypervisor [16]. Through meaningful experimental results, we found that our proposed TP-ARM scheme and SAWP method provide significant energy savings while guaranteeing acceptable performance required by users in practice. We note that that some part of this section is composed based on [17].

2.3.2 System Architecture

In this subsection, we discuss the architecture of the automated ACRB system. Our considered cloud environment including ACRB, which supports deploying a

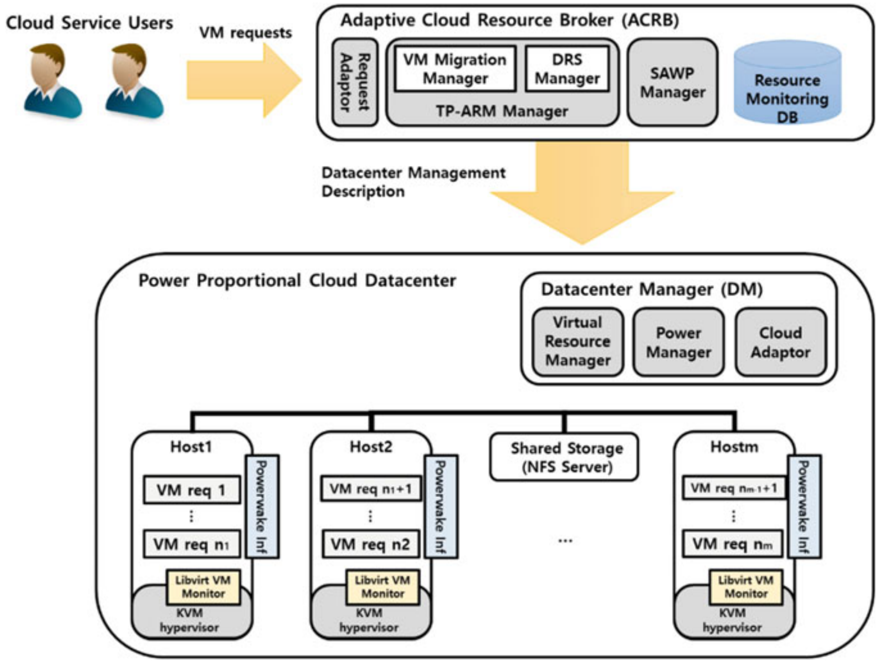


Fig. 2.4 Adaptive Cloud Resource Brokering (ACRB) system including TP-ARM module and SAWP manager for green cloud datacenters [17]

resource management scheme in order to migrate VM requests and adjusts the number of active servers according to the workload level, is depicted in Fig. 2.4. There are *m* physical hosts and *n* VM requests in the cloud datacenter. In the cloud datacenter, the information on resource utilization is collected through KVM hypervisor based monitoring modules into the Resource Monitoring DB module, and reported to the ACRB which is responsible for solving the migration of allocated VM requests and sizing the datacenter. The ACRB has two modules: the TP-ARM module and the SAWP Manager. The TP-ARM module includes the VM Migration Manager and the DRS Manager. In the first phase, the VM Migration Manager is responsible for selecting the appropriate VM requests to be migrated based on the measured resource utilization level. We describe the metrics for determining the VM request migration in detail in Sect. 2.2. In the second phase, the DRS Manager is responsible for finding the optimal number of active servers in the cloud datacenter. The DRS plan derived by the DRS Manager based on the amount of submitted VM requests and the measured resource utilization from each VM request is delivered to the Datacenter Manager, and the determined percentage of idle servers are powered off. The SAWP Manager is responsible for adjusting the window size of historical data adaptively in order to predict the future demand of VM requests. The SAWP Manager is able to achieve the exact prediction of future demands even under varied workload levels by considering the periodicity and the fluctuation of the historical data. The owner of the cloud datacenter has to minimize the costs for resource operations while boosting the benefits which can increase based on the good reputation of the observed QoS of the cloud services. In this paper, our ACRB tries to find a resource management solution to minimize the total cost of resource operations including two sub cost models: the energy consumption cost and the performance reputation cost.

From the perspective of the energy consumption cost, the VM Migration Manager tries to maximize the resource utilization of each physical host by consolidating running VM requests based on the whole offered load measured through the Libvirt VM Monitor module attached to each host. The Libvirt VM Monitor module gauges the utilization of resources such as CPU, memory, and I/O bandwidth in order to check whether whole hosts are overloaded [18, 19]. VM requests on overloaded hosts are preferably migrated to other hosts which have enough extra resource capacity to accommodate newly allocated VM requests. The DRS Manager sends the shutdown messages to servers which have to be powered off, while it sends magic packets to the ones which have to be powered on again. Obviously, the transition of servers from sleeping mode to active mode (aWake transition) requires additional energy consumption (note that the transition overhead from active to sleep (aSleep transition) can be negligible because it requires only a short time to be carried out compared to the aWake transition). Therefore, it is clear that frequent aWake transitions have to be discouraged in order to minimize unnecessary energy consumption. To simplify our model, we assume that the aSleep transition causes no additional energy consumption (in fact, it requires non-zero energy consumption). In terms of the performance reputation cost, the VM Migration Manager tries to decentralize running VM requests over multiple physical hosts in order to avoid QoS deterioration caused by VM interference. In cloud datacenters, the VM co-location interference is the key factor that makes servers undergo severe performance degradation [20, 21]. VM co-location interference is caused by resource contention which is reflected mainly by the number of co-located VM instances and the resource utilization of them. Briefly, VM co-location interference becomes larger as more VM instances are co-located on a common server, and subsequently, higher resource utilization occurs. Therefore, VM requests have to be scattered in order to avoid performance degradation by VM co-location interference as best as possible. Because of the complexity of the optimization for resource management in large-scale cloud datacenters, our TP-ARM scheme adopts a metaheuristic based on GA to obtain a near optimal solution for VM migration to achieve energy savings and QoS assurance in a cloud datacenter.

2.3.3 Cost Optimization Model of TP-ARM in Clouds

In this subsection, we introduce an energy cost model and performance reputation cost model based on our proposed TP-ARM scheme including VM live migration and DRS execution in the ACRM.

Workload cost model for phase 1: VM migration.
In general, VM requests have heterogeneous workloads in cloud datacenters. There are three types of VM request workloads: CPU, block I/O, and network I/O intensive workloads. The CPU intensive workloads such as scientific applications, high-definition video compression or big data analysis should be processed within an acceptable completion time determined by the cloud service users. The block I/O intensive workloads such as huge engineering simulations for critical areas include astrophysics, climate, and high energy physics which are highly data intensive [22]. These applications contain a large number of I/O accesses where large amounts of data are stored to and retrieved from disks. Network I/O intensive workloads such as Internet web services and multimedia streaming services have to be processed within a desirable response time according to their application type [18, 23]. Each VM request requires different resource utilizations according to their running applications. We categorized the resource utilization of running VM requests on a physical host in a cloud datacenter into two parts in this paper, the Flavor Utilization (FU) and the Virtual Utilization (VU). FU represents the ratio of the resource flavor (i.e., the specification of the resource requirement) of a VM request to the resource capacity of the physical host. VU represents the resource utilization of an assigned virtual resource. The Flavor Utilization $FU_{j,i}^{k}$ of a VM request i on a physical host j in the resource component k and the Virtual Utilization $VU_{j,i}^{k}$ of the VM request i on the physical host j in the resource component k are given by

$$FU_{j,i}^{k} = \frac{flv_{i}^{k}}{rcp_{j}^{k}} \tag{2.6}$$

$$RU_{j,i}^{k} = FU_{j,i}^{k} \cdot VU_{j,i}^{k} \tag{2.7}$$

where flv_{i}^{k} is the flavor of the VM request i in the resource component k; rcp_{j}^{k} is the resource capacity of the physical host j in the resource component k, and $RU_{j,i}^{k}$ is the actual resource utilization of the VM request i on the physical host j in the resource component k; therefore, it describes the practical workload in the resource component k. Note that the FU of the VM request can be determined through its attached service description to the ACRM in advance, while the VU can only be measured by the internal monitoring module in the cloud server during the running duration of the VM request. In the period t, the VM migration plan is designed according to the FU of the submitted VM requests at period t and the measured RU of each host in the past history $t-1$. The VM migration plan should satisfy the following constraints:

$$flv_{i}^{k} \leq S_{k}^{t}(i), \quad \forall i, k, t \tag{2.8}$$

$$flv_{i}^{k} \geq 0, \quad \forall i, k \tag{2.9}$$

$$S^{t}_{state}(i) = 1, \quad \forall i, t \tag{2.10}$$

where $S^{t}_{k}(i)$ denotes the remain capacity of resource k in the physical server S which is assigned to the VM request i at time period t, and $S^{t}_{state}(i)$ represents the state of the physical server S which is assigned to the VM request i at time period t. If the server S is in the sleep mode, then $S^{t}_{state}(i) = 0$, otherwise, $S^{t}_{state}(i) = 1$ (i.e., it is in the active mode). Equation (2.8) represents that the total requirements of the resource capacity of the newly allocated VM requests and migrated VM requests at time period t cannot exceed the resource capacity provided by their assigned physical server S. Next, we consider a VM performance reputation which is determined based on the RU of each physical server. The VM co-location interference implies that the virtualization of the cloud supports resource isolation explicitly when multiple VM requests are running simultaneously on a common PM however, it does not mean the assurance of performance isolation between VM requests internally. There is a strong relationship between VM co-location interference and both of the number of co-located VMs and their resource utilization in the PM. As the number of co-located VM requests increase and the RU of the VM requests becomes larger, VM co-location interference becomes more severe. The VM Migration Manager selects server candidates to be migrated based on the amount of RU for each physical server. To achieve this, the RU size of each server is measured through an internal monitoring module, and the VM Migration Manager checks whether they exceed the predetermined RU threshold value RU_{thr}. In servers which have an RU size over the RU_{thr}, they are considered as candidates for migration servers during the next period. After the migration servers are chosen, the VM requests to be migrated and their destination servers are determined based on the performance reputation cost model. We propose the objective function of the performance reputation cost C^{t}_{repu} at time C^{t}_{repu} given by

$$\begin{aligned} C^{t}_{repu} = {} & \rho_{intf} \sum_{\forall j} \sum_{\forall k} \omega_k \left|\mathcal{PM}^{t}_{j}\right| \cdot \left(\frac{\sum_{i=1}^{\left|\mathcal{PM}^{t}_{j}\right|} RU^{k}_{idx^{t}_{j,i}}}{RU^{k}_{thr}} - 1 \right)^{+} \\ & + \rho_{mig} T_{mig} \sum\nolimits_{\forall j} \left| \left|\mathcal{PM}^{t-1}_{j}\right| - \left|\mathcal{PM}^{t}_{j}\right| \right| \end{aligned} \tag{2.11}$$

where $(x)^{+} = \max(x, 0)$, and ρ_{intf} and ρ_{mig} are the price constants of the VM interference cost and VM migration cost, respectively. We use T_{mig} to denote the processing time for the VM migration. The first term in the right hand of Eq. (2.11) represents the following: as the number of concurrent running VM requests on a physical server increases, the number of users experiencing undesirable performance degradation also increases. The second term represents the following: as the number of migrated VM requests increases, the migration overhead is also increases. Therefore, a migration plan needs to be found that satisfies both avoiding unnecessary migration overhead and minimizing performance reputation degradation.

Energy Consumption Cost Model for Phase 2: DRS Procedure.
To achieve a power-proportional cloud datacenter which consumes power only in proportion to the workload, we considered a DRS procedure which adjusts the number of active servers by turning them on or off dynamically [6]. Obviously, there is no need to turn all the servers in a cloud datacenter on when the total workload is low. In the DRS procedure, the state of servers which have no running applications can be transited to the power saving mode (e.g., sleep or hibernation) in order to avoid wasting energy. At each time *t*, the number of active servers in the cloud datacenter is determined by a DRS procedure plan according to the workload of the allocated VM requests. In order to successfully deploy the DRS procedure onto our system, we considered the switching overhead for adjusting the number of active servers (i.e., for turning servers in sleep mode on again). The switching overhead includes the following: (1) additional energy consumption from the transition from a sleep to active state (i.e., awaken transition); (2) wear-and-tear cost of the server; (3) fault occurrence by turning servers in sleep mode [6]. We considered the energy consumption as the overhead from DRS execution. Therefore, we define the constant P_{aWake} to denote the amount of energy consumption for the aWake transition of the servers. Then, the total energy consumption C^t_{energy} of the cloud datacenter at time *t* is given by

$$C^t_{energy} = \rho_p P_{active} \sum\nolimits_{\forall j} \left(\left| \mathcal{PM}^t_j \right| \right)^- + \rho_p P_{aWake} T_{switch} \left(\sum\nolimits_{\forall j} \left(\left| \mathcal{PM}^t_j \right| \right)^- - \sum\nolimits_{\forall j} \left(\left| \mathcal{PM}^t_j \right| \right)^- \right)^+ \tag{2.12}$$

where $(x)^- = \min(x, 1)$. We use ρ_p to denote the constant of the power consumption price, and P_{active} and P_{aWake} are the amount of power consumption for the active mode and the aWake transition of the server, respectively. T_{switch} is the time requirement for the aWake transition. The first term on the right side of Eq. (2.12) represents the energy consumption for using servers to serve VM requests allocated to all the physical servers in the cloud datacenter at time *t*, and the second term represents the energy consumption for the awaken transition of sleeping servers. Especially, the second term implies that a frequent change in the number of active servers could increase an undesirable waste of energy. Note that the overhead by the transition from the active to the sleep state (i.e., asleep transition) is ignored in our model because the time required for the asleep transition is relatively short compared to the one for the awaken transition.

Algorithm 2. Phase 1: VM migration in TP-ARM

```
Input : the VM requests allocation of each server $\mathcal{PM}^{t-1}, \forall j$
Output : the VM migration plan $\mathcal{PM}^{t}, \forall j$
for each $\mathcal{PM}_j^{t-1} \in \mathcal{PM}^{t-1}, \forall j$
    for each $VM_{idx_{j,i}^{t-1}}, \forall i$
        for each resource component $k$
            measure resource utilization $RU_{idx_{j,i}^{t-1}}^{k}$ of the VM request with $idx_{j,i}^{t-1}$
        end for
    end for
end for
calculate the average resource utilization $\overline{RU^k}$ at all resource components $k$
if $\overline{RU^k} < RU_{thr^{low}}^{k}, \forall k$ then
    derive the VM migration plan $\mathcal{PM}^{t}$ based on Eq.(8) through Algorithm 3.
else if $\overline{RU^k} > RU_{thr^{high}}^{k}$ then
    go to Algorithm 2.
end if
```

The purpose of our algorithm is to achieve a desirable VM migration and DRS procedure plan that are energy efficient as well as a QoS aware resource management at each period iteratively. At period $t - 1$, the proposed TP-ARM approach aims to minimize the cost function in Eq. (2.8) by finding the solution $\mathcal{PM}^t$ as follows,

$$minimize_{\mathcal{PM}^t} \quad C_{total}^{t} = C_{repu}^{t} + C_{energy}^{t} \tag{2.13}$$

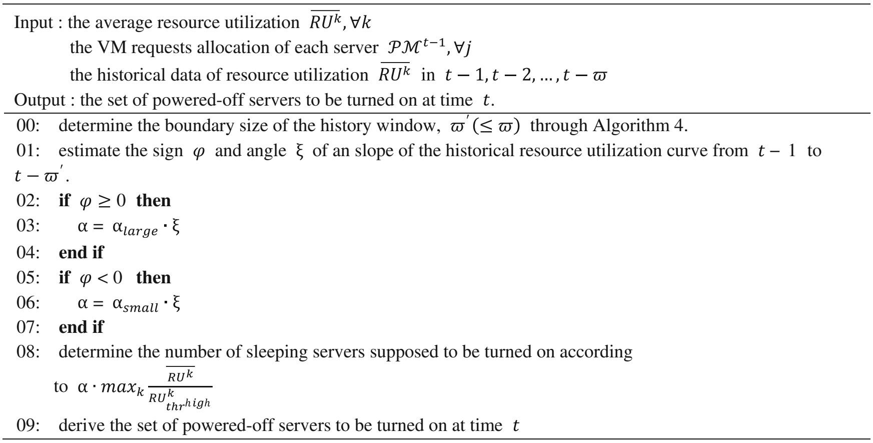
Subject to Eqs. (2.8–2.10)

To solve the objective cost function in Eq. (2.13), we prefer a well-known evolutionary metaheuristic called the Genetic Algorithm in order to approximate the optimal plan $\mathcal{PM}^t$ at each period because Eqs. (2.11) and (2.12) have non-linear characteristics. In next section, our proposed TP-ARM with the VM migration and DRS procedure is introduced in detail.

2.3.4 Heuristic Algorithms for the Proposed TP-ARM Scheme

In this subsection, we describe heuristic algorithms for the proposed TP-ARM scheme discussed in Algorithms 2, 3, and 4. First, the process of VM migration in the TP-ARM approach includes the following four steps: (1) monitor and collect the resource utilization data on VM requests for each physical server through the attached Libvirt based monitoring tools; (2) go step 3 if the average utilization of all the active servers are significantly low (i.e., below the predefined threshold), otherwise go to step 4; (3) choose active servers which are supposed to be turned off, migrate all the VM instances on them to other servers and trigger DRS execution; (4) determine the number of sleeping servers which are supposed to be turned on and send magic packets to them for wake-up if the average utilization of all the active servers are significantly high (i.e., above the predefined threshold), otherwise maintain the current number of active servers. Algorithm 2 shows the procedure for Phase 1:VM migration in the TP-ARM scheme. From line 00 to 06, the resource utilization RU of all the VM requests on the physical servers in the cloud datacenter is measured by the Libvirt API monitoring module. The average resource utilization $\overline{\mathrm{RU}^{\mathrm{k}}}$ at all resource components k is calculated in line 07. If the $\overline{\mathrm{RU}^{\mathrm{k}}}$ is below the predetermined $\mathrm{RU}^{\mathrm{k}}_{\mathrm{thr}^{\mathrm{low}}}$ at all the resource components, then the optimal VM migration plan $\mathcal{PM}^{\mathrm{t}}$ is derived, and the algorithm is finished. Otherwise, if the $\overline{\mathrm{RU}^{\mathrm{k}}}$ exceeds the predetermined $\mathrm{RU}^{\mathrm{k}}_{\mathrm{thr}^{\mathrm{high}}}$, then Algorithm 2 for the DRS procedure is triggered.

Algorithm 3. Phase 2: DRS procedure in TP-ARM



```
Input : the average resource utilization $\overline{RU^k}, \forall k$
        the VM requests allocation of each server $\mathcal{PM}^{t-1}, \forall j$
        the historical data of resource utilization $\overline{RU^k}$ in $t-1, t-2, \ldots, t-\varpi$
Output : the set of powered-off servers to be turned on at time $t$.
00: determine the boundary size of the history window, $\varpi' (\le \varpi)$ through Algorithm 4.
01: estimate the sign $\varphi$ and angle $\xi$ of an slope of the historical resource utilization curve from $t-1$ to
    $t-\varpi'$.
02: if $\varphi \ge 0$ then
03:     $\alpha = \alpha_{large} \cdot \xi$
04: end if
05: if $\varphi < 0$ then
06:     $\alpha = \alpha_{small} \cdot \xi$
07: end if
08: determine the number of sleeping servers supposed to be turned on according
    to $\alpha \cdot max_k \frac{\overline{RU^k}}{RU^k_{thr^{high}}}$
09: derive the set of powered-off servers to be turned on at time $t$
```

Algorithm 3 shows the procedure for Phase 2: DRS in the TP-ARM scheme. In line 00, the history window size ϖ' is determined by the ACRM through Algorithm 4: the SAWP scheme that is described in next section. In line 02, we calculate the sign φ and the angle ξ of the slope of the historical resource utilization curve from the current time $\mathrm{t}-1$ to the time $\mathrm{t}-\varpi'$. If $\varphi \geq 0$ (i.e., the resource utilization is increased), and then, we get the coefficient α based on $\alpha_{\mathrm{large}} \cdot \xi(\alpha_{\mathrm{large}} > \alpha_{\mathrm{small}})$ to adaptively react to the large workload level in the cloud datacenter. Otherwise, if $\varphi < 0$ (i.e., the resource utilization is decreased), then we get α based on $\alpha_{\mathrm{small}} \cdot \xi$ to maximize the energy saving of the cloud datacenter. In line 08, we determine the number of servers to be in active mode in the next period.

Algorithm 4 describes the GA for the TP-ARM scheme in detail. $\mathrm{pop}^{\mathrm{h}}$ is a population with size P (even number) at the hth generation. $\mathrm{h}_{\mathrm{max}}$ is the maximum of the GA iteration count. In line 03, $\mathrm{f}(\cdot)$ is a fitness function of the total cost with two parameters, candidate solution $\mathcal{PM}^{\mathrm{h,i}}$ and the previous solution $\mathcal{PM}^{\mathrm{t-1}}$ at time $\mathrm{t}-1$. From line 04 to 06, the two candidate solutions are iteratively chosen randomly from pop to generate offsprings by crossover until there are no remaining unselected solutions in pop. From line 07 to 08, the fitness function values of each offspring are calculated similar to line 03. In line 09, all the parent solutions in pop and generated offsprings are sorted in ascending order for their corresponding fitness function values. In line 10, the next population including only P solutions that achieve good performance from the union of the original pop and derived offsprings is generated to improve the quality of the final solution. From line 11 to 12, to reduce the time complexity of the GA procedures, when we encounter the first solution that has a fitness function value below the predetermined fitness threshold value fv_{thr}, it counts as a final solution for the next period, and the algorithm is finished. In line 14, if we cannot find a solution that satisfies fv_{thr} when the iteration count reaches h_{max}, then we select a solution that has a minimum fitness function value in the population which satisfies the conditions in Eqs. (2.8–2.10) as a final solution for the next period. If there are no solutions to satisfy all the constraints, then we just preserve the current resource allocation vector shown from line 15 to 16. In the population of a GA, mutations are often applied in order to include new characteristics from the offsprings that are not inherited traits from the parents [24]. We did not consider mutations in our GA in this paper; however, it can be used to improve the quality of the GA for the TP-DRM in future work.

Algorithm 4. GA for searching VM migration plan in TP-ARM

```
Input : the set of VM requests allocation of whole servers $\mathcal{PM}^{t-1}$ at time $t-1$
Output : the set of VM requests migration plan, $\mathcal{PM}^{t}$ at time $t$
00: initialize $\boldsymbol{pop}^h = \{\mathcal{PM}^{h,1}, \mathcal{PM}^{h,2}, \ldots, \mathcal{PM}^{h,P}\}$, $P = size\ of\ population$,
    and even number
01: while $h \le h_{max}$
02:     for each $\mathcal{PM}^{h,i} \in \boldsymbol{pop}^h$
03:         $fv^{h,i} = f_{C_{total}^{t}}(\mathcal{PM}^{h,i}, \mathcal{PM}^{t-1})$
04:     while $!\,(\forall \mathcal{PM}^{h,i} \in \boldsymbol{pop}^h\ are\ selected\ as\ parents)$
05:         $(\mathcal{PM}^{h,i}, \mathcal{PM}^{h,j}) \leftarrow$ select parents randomly from $\boldsymbol{pop}^h$
06:         $\boldsymbol{offspring}^h =\cup$ crossover $(\mathcal{PM}^{h,i}, \mathcal{PM}^{h,j})$
07:     for each $off\_\mathcal{PM}^{h,i} \in \boldsymbol{offspring}^h$
08:         $off\_fv^{h,i} = f_{C_{total}^{t}}(off\_\mathcal{PM}^{h,i}, \mathcal{PM}^{t-1})$
09:     sort $\boldsymbol{pop}^{h\sim} = \left\{\mathcal{PM}^{h,i}, \ldots, \mathcal{PM}^{h,P}, off\_\mathcal{PM}^{h,1}, \ldots, off\_\mathcal{PM}^{h,\frac{P}{2}},\right\}$ in
        ascending order of solutions' corresponding $fv^{h,i}\ and\ off\_fv^{h,i}$
10:     $\boldsymbol{pop}^{h+1} = \{\boldsymbol{pop}_1^{h\sim}, \ldots, \boldsymbol{pop}_P^{h\sim}\}$
11:     if $f_{C_{total}^{t}}(\boldsymbol{pop}_1^{h+1}, \mathcal{PM}^{t-1}) \le fv_{thr}$ then
12:         $\mathcal{PM}^t = \boldsymbol{pop}_1^{h+1}$ and exit
13:     $h++$
14:     $\mathcal{PM}^t = \mathrm{argmin}_{\boldsymbol{pop}_i^{h_{max}} \in \boldsymbol{pop}^{h_{max}}} \left(f_{C_{total}^{t}}\left(\boldsymbol{pop}_i^{h_{max}}, \mathcal{PM}^{t-1}\right)\right)$ s.t. Eq (3), (4), (5)
15: if $\mathcal{PM}^t = \emptyset$ then
16:     $\mathcal{PM}^t = \mathcal{PM}^{t-1}$
```

Self-Adjusting Workload Prediction Scheme. Algorithm 5 describes the procedure of the proposed SAWP algorithm in ACRB. The irregularity function $\boldsymbol{g}(\boldsymbol{\varepsilon}, \boldsymbol{\delta})$ represents a level of unpredictability for future resource utilization where ε is its fluctuation value (i.e., levels of instability and aperiodicity), and δ is its burstiness value (i.e., levels of sudden surge and decline), and both values are calculated based on [15]. According to the predetermined threshold values $\boldsymbol{g}_{thr}^{high}$ and $\boldsymbol{g}_{thr}^{low}$ with $\boldsymbol{g}(\boldsymbol{\varepsilon}, \boldsymbol{\delta})$, the history window size ϖ' is adaptively updated at each prediction process. A relatively long history window size is not suitable to react to recent changes in the workload but is tolerant to varied workload patterns over a short time while a short history window size is favorable to efficiently respond to the latest workload patterns but is not good for widely varying workloads. Consequently, the SAWP algorithm generally outperforms traditional prediction schemes during drastic utilization changes from various cloud applications because it is able to cope with temporary resource utilizations (i.e., does not reflect overall trends) by adjusting the history window size ϖ'.

Algorithm 5. Self-Adjusting Workload Prediction scheme

```
Input : the historical data of resource utilization $\overline{RU^k}$ in $t-1, t-2, \ldots, t-\varpi$
Output : the boundary size of history window $\varpi'$

$\varpi' = \varpi$
analyze the historical data of resource utilization $\overline{RU^k}$ from $t-1$ to $t-\varpi$
extract the fluctuation size $\varepsilon$ and the burstness value $\delta$ based on [6]
if $g(\varepsilon,\delta) > g_{thr}^{high}$ then
    increase $\varpi'$ based on $\frac{g(\varepsilon,\delta)}{g_{thr}^{high}}$
end if
if $g(\varepsilon,\delta) < g_{thr}^{low}$ then
    decrease $\varpi'$ based on $\frac{g_{thr}^{low}}{g(\varepsilon,\delta)}$
end if
predict the future demand based on $\overline{RU^k}$ from $t-1$ to $t-\varpi'$
```

2.3.5 Evaluation

In our experiments, we measured various metrics which affect the parameter decision for our proposed algorithms. To this end, we established five cluster servers as a cloud platform, one server for ACRB with a Mysql DB system, a power measuring device from Yocto-Watt [25] and a laptop machine called the VirtualHub for collecting and reporting information on the measured power consumption shown in Fig. 2.5. The hardware specification of each server for the cloud compute host is as follows: an Intel i7-3770 (8-cores, 3.4 Ghz), 16 GB RAM memory, and two 1 Gbps NICs (Network Interface Cards). In order to measure efficiently the power consumption of the cloud cluster server, we used a power measuring device model called YWATTMK1 by Yocto-Watt. This model has a measurement unit of 0.2 W for AC power with an error ratio of 3% and 0.002 W for the DC power with an error ratio of 1.5%. The VirtualHub collects information

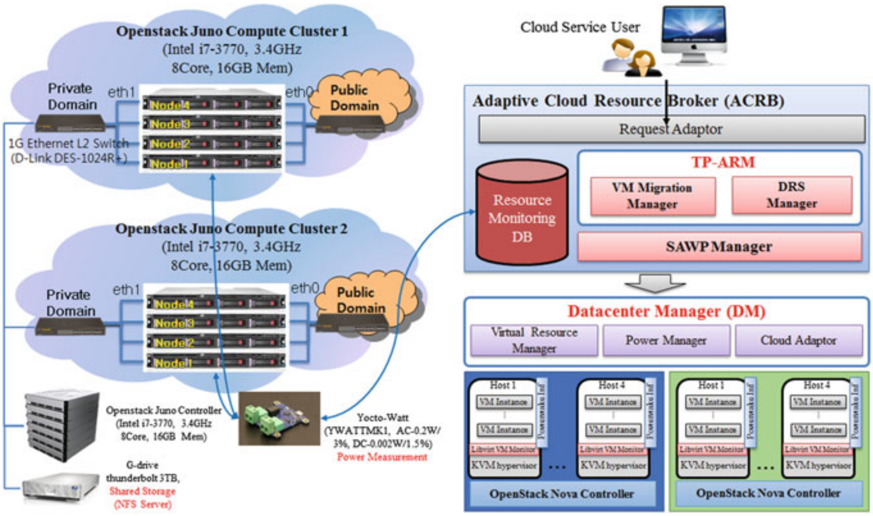


Fig. 2.5 Experimental environment

on power consumption from YWATTMK1 through the Yocto-Watt Java API and reports it to the power monitoring table of the Mysql DB system in the ACRB periodically.

The dynamic resource utilizations by each VM instance are measured via our developed VM monitoring modules based on the Libvirt API and are sent to the resource monitoring table of the Mysql DB system periodically. In addition, a SATA 3 TB hard disk called the G-drive was deployed as a NFS server in our testbed for live migration [26]. We adopted Openstack kilo version which is a well-known open source solution based on KVM Hypervisor as a cloud platform in our testbed. Finally, we used PowerWake package [27] to turn remotely off servers on via Wake on Lan (WOL) technology for DRS execution.

In Table 2.1, we show the average power consumption and resource utilization of two running applications: Montage m106-1.7 projection and ftp transfer. Montage project is an open source based scientific application, and it has been invoked by NASA/IPAC Infrared Science Archive as a toolkit for assembling Flexible Image Transport System (FITS) images into custom mosaics [10, 28]. The m106-1.7 projection in Montage is a CPU-intensive application while the ftp transfer is a network-intensive one. Therefore, the running of m106-1.7 causes a power consumption of about 75 Wh and a CPU utilization of about 15% whereas the power consumption by ftp transfer for a 1.5 GB test.avi file is about 60 Wh, and the network bandwidth usage is about 3.7 Mbps. That is, the CPU usage is the main part that affects the power consumption of server. In terms of DRS execution, the power consumption by an off server is about 2.5 Wh (note that this value is not zero because the NIC and some its peripheral components are still powered on to maintain the standby mode to receive the magic packets from the Powerwake

Table 2.1 The average power consumption and resource utilization of two running applications: Montage m106-1.7 projection and ftp transfer

Host state		Power consumption (Wh)	CPU utilization	Mem utilization	Net bandwidth
Idle		55 Wh	1.5%	3%	20 Kbps (bytes)
Active	Montage m 106-1.7 (projection)	75 Wh	15%	3.7%	20 Kbps (bytes)
	test.avi downloading (ftp, 1.5 GB)	60 Wh	2%	3.7%	3.7 Mbps (bytes)
aSleep (to Power Off)		70–80 Wh (5–7 s)	1.8% (5–7 s)	3% (5–7 s)	20 Kbps (bytes)
Power off (hibernating)		2.5 Wh			
aWake (from power off)		78 Wh (50 s–1 min)			

controller) while the asleep and awaken transition procedures, which cause the switching overhead for DRS, require power consumption of about 80 Wh to turn the active servers off or to turn off servers on, respectively. The asleep transition procedure is trivial because it requires a short time (i.e., 5–7 s) to complete even though its power consumption is considerable whereas the awaken transition procedure requires a relatively long execution time (i.e., more than 1 min) and should be considered carefully. The overhead for the awaken transition would be a more serious problem in practice because its required execution time is generally far longer (i.e., above 10 min) for multiple servers of racks in a datacenter. Therefore, it is essential to consider the switching overhead for awaken transition to reduce efficiently the resource usage cost of the datacenter.

Figure 2.6 shows the additional energy consumption overhead of a single physical server from the DRS execution with respect to the asleep transition (i.e., from the active mode to the sleep mode) and the awaken transition (i.e., from the sleep mode to the active mode). Note that the asleep transition requires a relatively short processing time of about 7 or 8 s and causes an additional energy consumption of about 20% compared to the idle state of the server, while the awaken transition needs 60 or 90 s to be carried out. Although not included in this paper, we also verified that some of the HP physical servers in our laboratory required more than 10 min to be turned on. We expect that physical servers in a real cloud datacenter require tens of minutes or hundreds of minutes to get back from the sleep mode to the active mode. Moreover, the aWake transition causes an additional energy consumption of about 40% on average compared to the idle state of a server. These results imply that the frequent DRS execution might cause the degradation of the energy saving performance in a cloud datacenter. Our proposed TP-ARM scheme is able to avoid the unnecessary energy consumption overhead by the

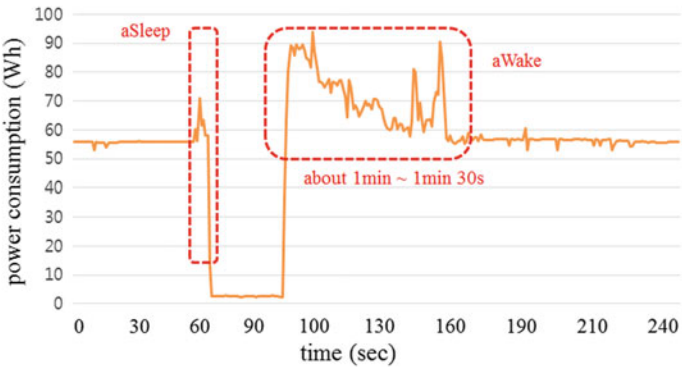


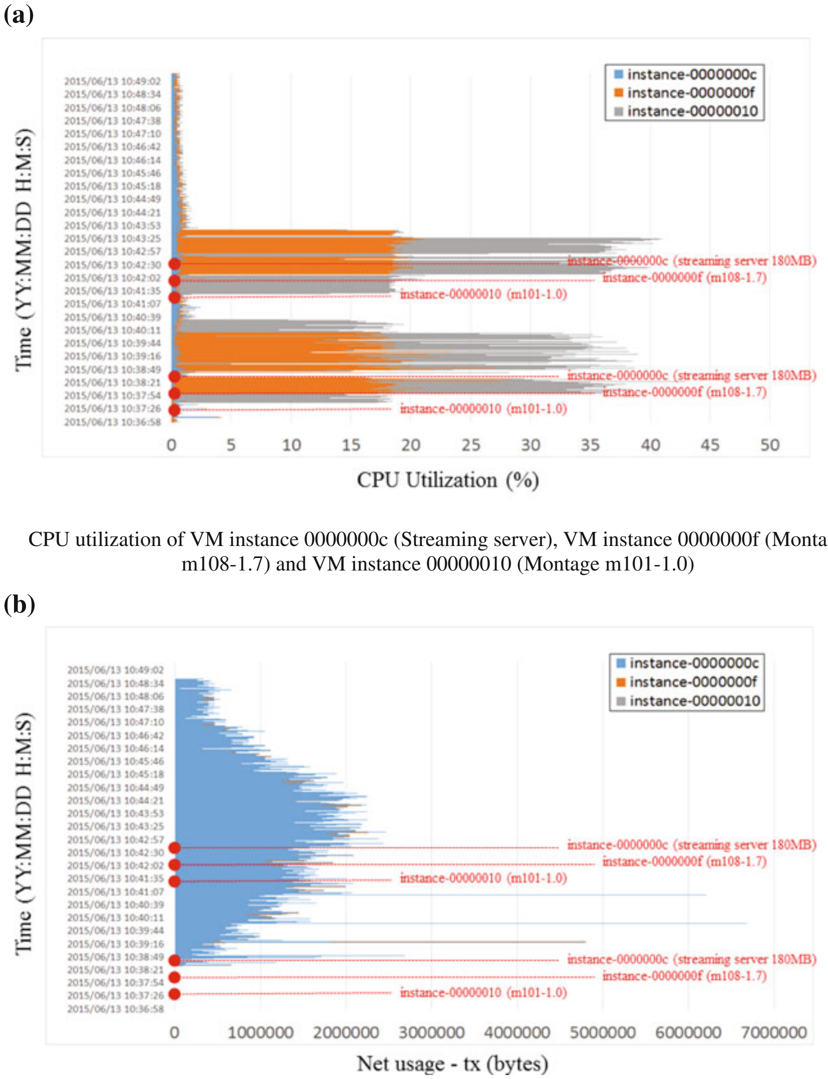
Fig. 2.6 Power consumption overhead from DRS execution of the test server with respect to the aSleep and aWake transition

awaken transition through the cost model by considering the DRS transition overhead shown in Eq. (2.12).

Figure 2.7 shows the resource utilization and the power consumption measured by the Libvirt API monitoring module and Yocto-Watt power measuring device. VM request instance-10 runs m101-1.0 mProj, instance-0f runs m108-1.7, and instance-0c runs the streaming server with the 180 MB movie file. Figure 2.7a–c shows that their resource utilization is consistent with the resource intensive characteristic of their running workload. Figure 2.7d shows the different energy consumptions according to each workload type. The energy consumption of the physical server is 60 Wh in the idle state; it is about 82 Wh when running m101-1.0 and 95 Wh when running both m101-1.0 and m108-1.7, while the streaming server just causes an additional energy consumption of about 2 Wh. As mentioned in Sect. 2.3.3, the main part of the resource components affecting the power consumption in a physical server is the CPU resource, and the effects of other components such as the memory, storage, and network interface cards are negligible in general. These results are consistent with the ones in Table 2.1. Therefore, these results imply that the energy consumption can be different according to which resource component has high utilization. Our proposed workload cost model of the TP-ARM scheme considers different weight values for each resource component in order to derive the practical cost of the resource management.

Figure 2.8 shows the performance of the CPU utilization for the VM live migration between the source machine (kdkCluster2) and the destination machine (kdkCluster1). There are VM request instances, such as running instance-33 and 34, which execute the compression of video files 5 and 4 GB in size, respectively.

Instance-33 executed the process of m101-1.6mProj at 12:40:14 local time and started migration to the kdkCluster1 at 12:44:00 local time. The migration of instance-33 was completed by 13:24:30 local time, and its execution of m101-1.6 mProj ended at 13:47:16. Namely, the completion time of m101-1.6 mProj at instance-33 was 67 min in total, and its migration times were over 30 min. If we

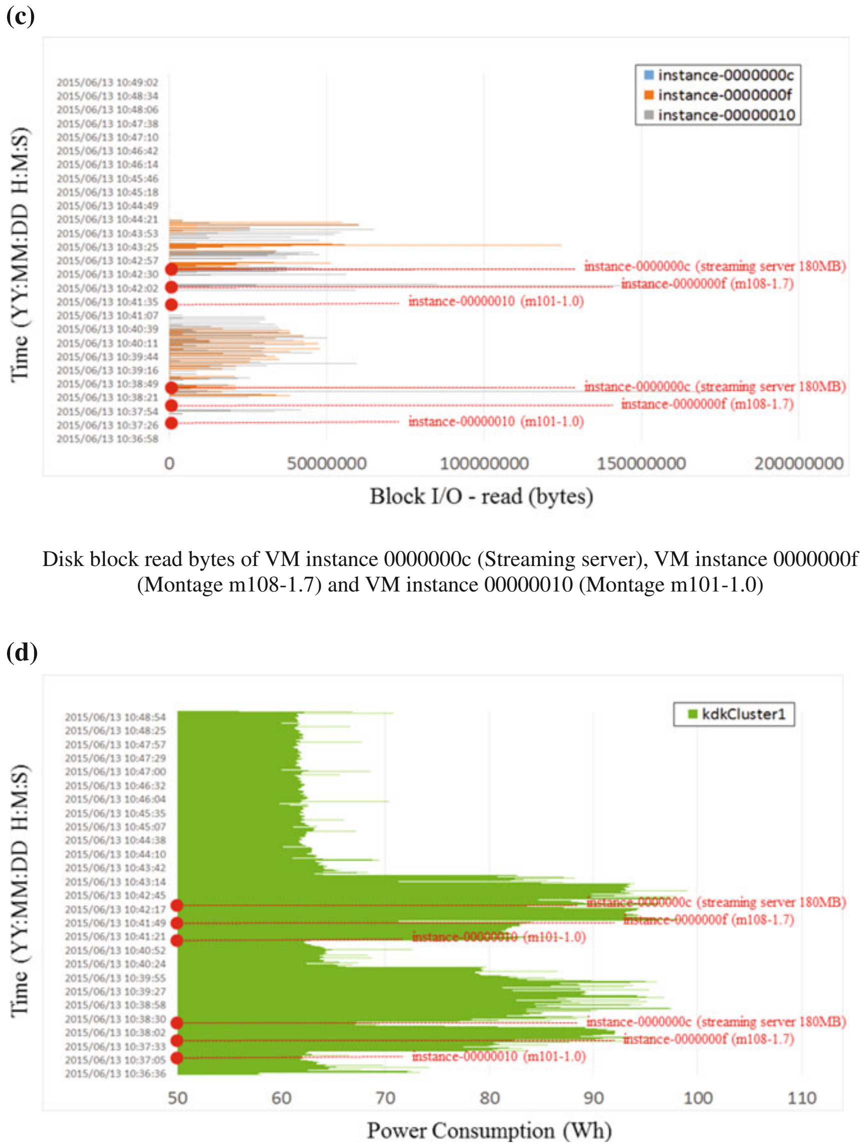


CPU utilization of VM instance 0000000c (Streaming server), VM instance 0000000f (Montage m108-1.7) and VM instance 00000010 (Montage m101-1.0)

Network transmission bytes of VM instance 0000000c (Streaming server), VM instance 0000000f (Montage m108-1.7) and VM instance 00000010 (Montage m101-1.0)

Fig. 2.7 Libvirt API monitoring module shows the results of resource utilization of running VM requests in the kdkCluster1 in CPU utilization (**a**) network transmission bytes (**b**)

consider the time to complete m101-1.6 mProj in the case of no VM live migration, it is forecasted to be around 10 min. We can see that there exist quite big overheads in VM live migration.



Disk block read bytes of VM instance 0000000c (Streaming server), VM instance 0000000f (Montage m108-1.7) and VM instance 00000010 (Montage m101-1.0)

Power consumption of VM instance 0000000c (Streaming server), VM instance 0000000f (Montage m108-1.7) and VM instance 00000010 (Montage m101-1.0)

Fig. 2.7 (continued). Libvirt API monitoring module shows the results of resource utilization of running VM requests in the kdkCluster1 in Disk block read bytes (**c**) and Yocto-Watt module [8] was used to measure the power consumption of the kdkCluster1 as shown in (**d**)

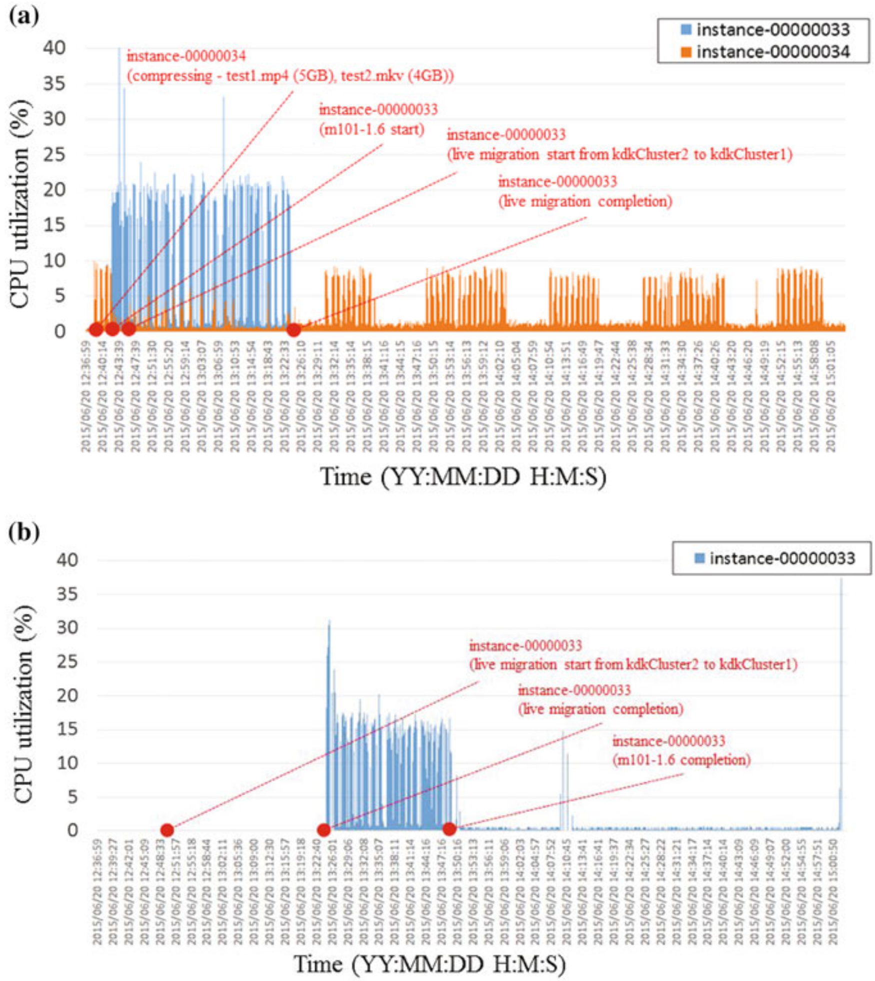
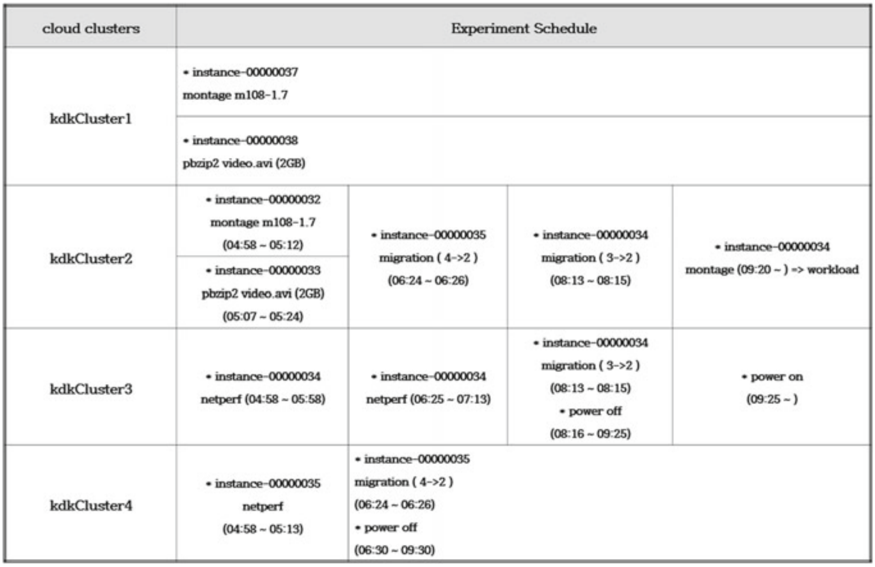


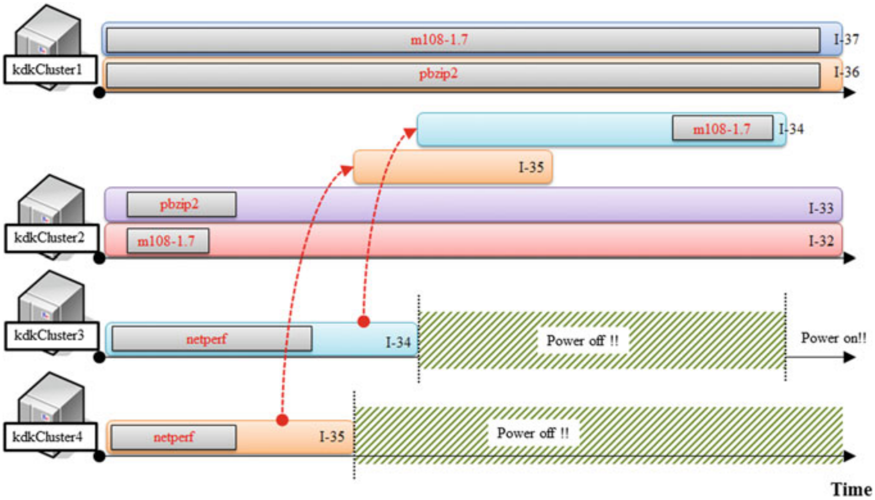
Fig. 2.8 Results of CPU utilization of the migrated VM request from **a** a source server: kdkCluster2 to **b** a destination server:kdkCluster1

In addition, VM migration shows some problems in power consumption. We considered cost models to identify an efficient migration scheme which was defined in Eq. (2.11). Figure 2.9 shows the test environments and the operation of the VM migration when the proposed TP-ARM schemes are applied to the test cloud systems with heterogeneous applications and test programsincluding m108-1.7, pbzip2, and netperf. From the experiments, we obtained interesting results in the comparison of the utilization performance, which examined the CPU utilization monitoring results and the measured power consumption, respectively, shown in Fig. 2.10. From the results, we found rapidly changing instants of utilization when the cloud brokering system considered the power consumptions for each running



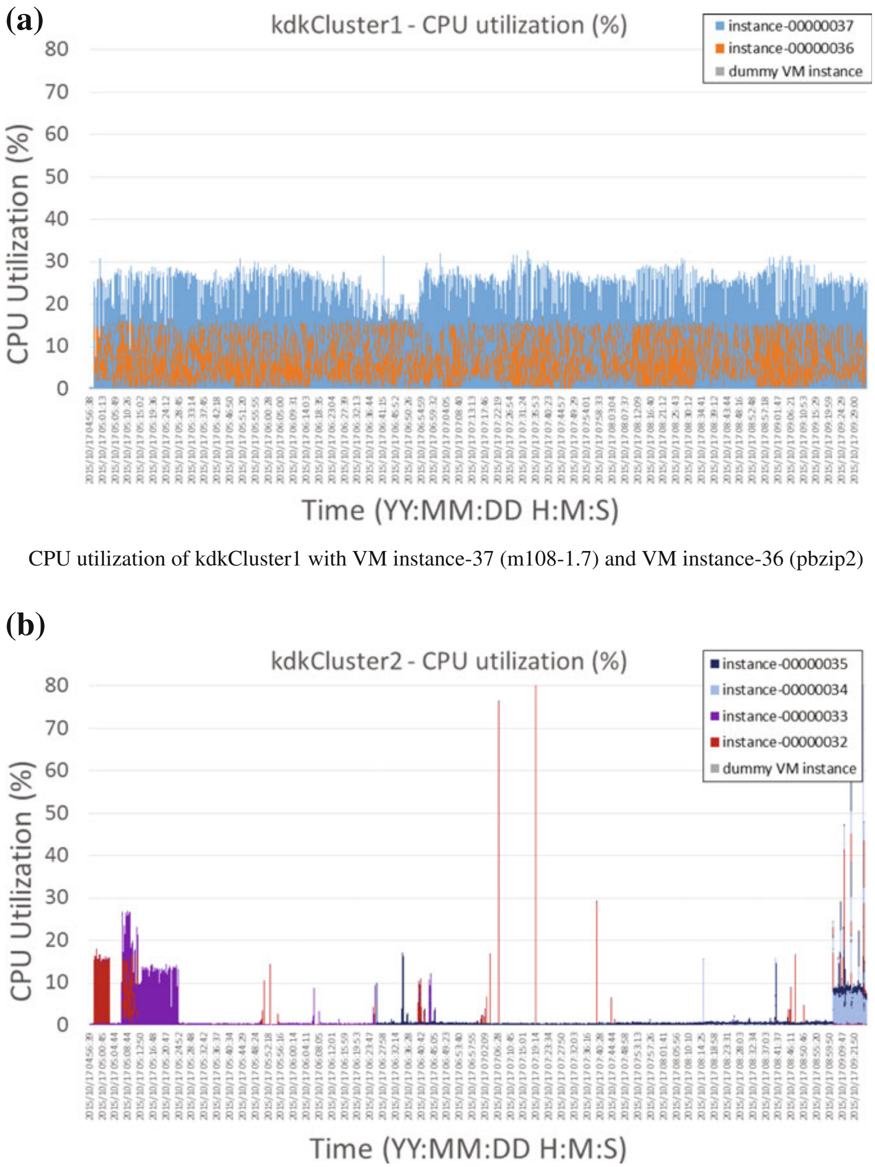
cloud clusters	Experiment Schedule			
kdkCluster1	• instance-00000037 montage m108-1.7			
	• instance-00000038 pbzip2 video.avi (2GB)			
kdkCluster2	• instance-00000032 montage m108-1.7 (04:58 ~ 05:12) • instance-00000033 pbzip2 video.avi (2GB) (05:07 ~ 05:24)	• instance-00000035 migration (4->2) (06:24 ~ 06:26)	• instance-00000034 migration (3->2) (08:13 ~ 08:15)	• instance-00000034 montage (09:20 ~) => workload
kdkCluster3	• instance-00000034 netperf (04:58 ~ 05:58)	• instance-00000034 netperf (06:25 ~ 07:13)	• instance-00000034 migration (3->2) (08:13 ~ 08:15) • power off (08:16 ~ 09:25)	• power on (09:25 ~)
kdkCluster4	• instance-00000035 netperf (04:58 ~ 05:13)	• instance-00000035 migration (4->2) (06:24 ~ 06:26) • power off (06:30 ~ 09:30)		

(a) The schedule of the use case



(b) The illustration of the use case

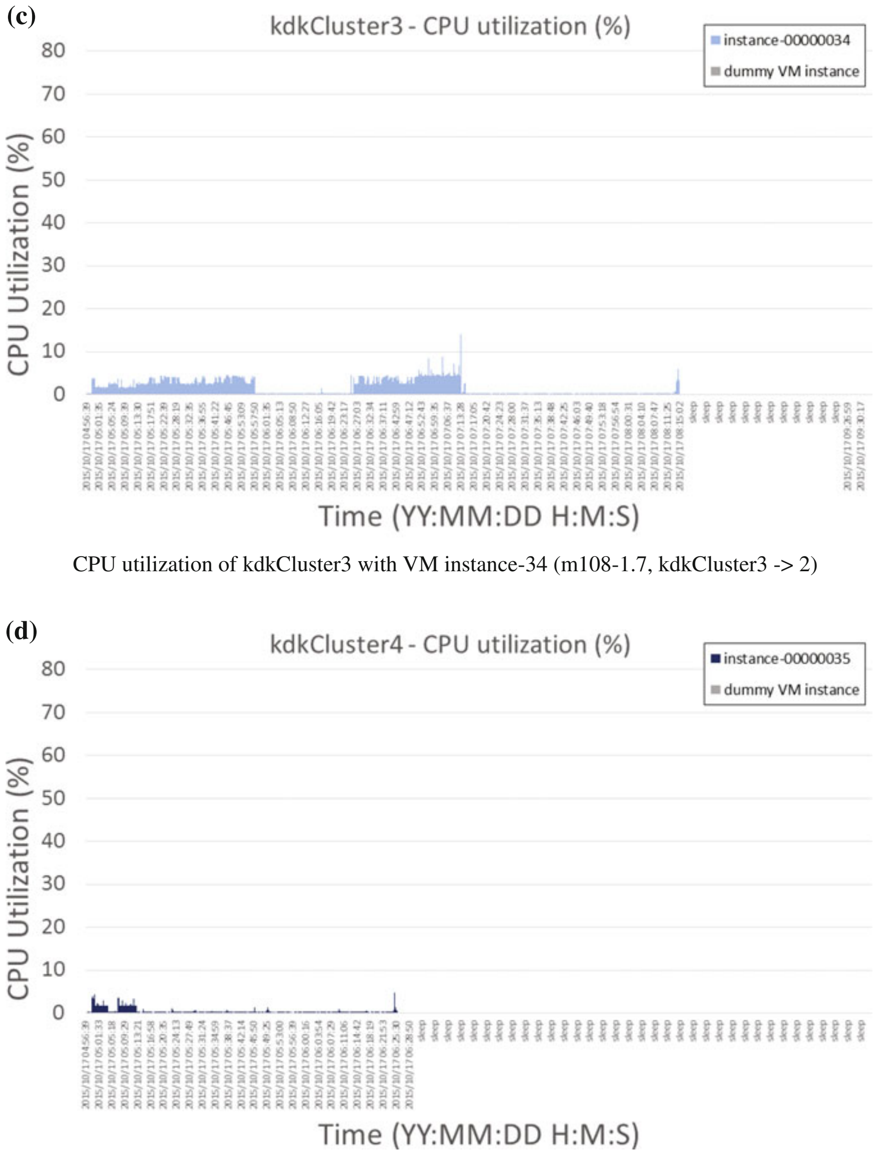
Fig. 2.9 Test environments and operation of the VM migration when the proposed TP-ARM schemes are applied for test cloud servers (e.g. kdkCluster1–5 in lab test environments) with heterogeneous applications and test programs, such as m108-1.7, pbzip2, and netperf. **a** An example of test schedule under cloud testbed environments. **b** Illustration of the VM migration using two-phase power consumption cost models



CPU utilization of kdkCluster1 with VM instance-37 (m108-1.7) and VM instance-36 (pbzip2)

CPU utilization of kdkCluster2 with VM instance-35 (netperf, kdkCluster4 -> 2), VM instance-34 (m108-1.7, kdkCluster3 -> 2), VM instance-33 (pbzip2) and VM instance-32 (m108-1.7)

Fig. 2.10 Performance comparison of CPU utilization using 4 test systems in lab environments. **a** and **b** show the results of the measured utilization during the VM live migration in test use case as shown in Fig. 2.9.



CPU utilization of kdkCluster3 with VM instance-34 (m108-1.7, kdkCluster3 -> 2)

CPU utilization of kdkCluster4 with VM instance-35 (netperf, kdkCluster4 -> 2)

Fig. 2.10 (continued). Performance comparison of CPU utilization using 4 test systems in lab environments. **c** and **d** show the results of the measured utilization during the VM live migration in test use case as shown in Fig. 2.9

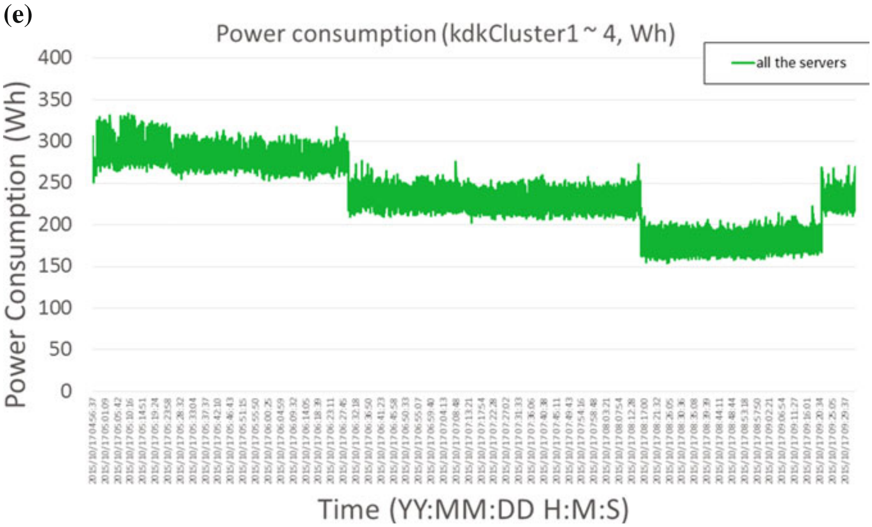


Fig. 2.10 (continued). Performance comparison of CPU utilization using 4 test systems in lab environments. **e** Measured power consumption of test cloud systems (e.g. kdkCluster1–4) in lab cloud test environments

application under cloud data center environments. Additionally, the test set 'netperf', a network intensive workload of a test cloud system (e.g., kdkCluster4 with instance-35) in Fig. 2.10d, showed very low performance in CPU utilization. It was enough to satisfy the threshold value to run the VM migration efficiently. And then, the proposed TP-ARM algorithm adjusts the VM migration procedure to reduce the power consumption sufficiently. Basically, the TP-ARM is triggered to migrate instance-35 to another system kdkCluster2; thereafter, it changes the kdkCluster's sleeping mode in advance.

Next, because instance-34 of the kdkCluster system was running netperf, which generated a very low performance of utilization, the TP-ARM also did a migration of the instance-34 to the kdkCluster2 and transited to the sleeping mode as well. Therefore, we could expect efficient power consumption through the change in the sleeping mode of kdkCluster3 and kdkCluster4, respectively, as well as keeping active modes for both kdkCluster1 and kdkCluster2. Figure 2.10 shows the performance comparison of the power consumption for the VM migration and DRS process based on the TP-ARM algorithm, e.g., seen in Fig. 2.9. We can see a performance improvement of 60 Wh each in power consumption after the change in the sleeping mode in the case of the VM migration requests. Through the experiments, we verified that the proposed TP-ARM scheme, which carries out adaptive migration and asleep transition through real-time monitoring of resource utilization, has good performance in reducing the power consumption effectively. This study provides good results for achieving an energy efficient cloud service for users by not increasing QoS degradation as much.

2.3.6 Conclusion

In this paper, we introduced ACRB with Two Phases based the TP-ARM scheme for energy efficient resource management by real time based VM monitoring in a cloud data center. Our proposed approach is able to reduce efficiently the energy consumption of the servers without a significant performance degradation by live migration and DRS execution through a considerate model that considers switching overheads. The various experimental results based on the Openstack platform suggest that our proposed algorithms can be deployed to prevalent cloud data centers. The novel prediction method called SAWP is proposed in order to improve the accuracy of forecasting future demands even under drastic workload changes.

Especially, we evaluated the performance of our proposed TP-ARM scheme through various applications such as Montage, pbzip2, netperf, and the streaming server which have heterogeneous workload demands. Our TP-ARM scheme could maximize the energy saving performance of the DRS procedure by Phase 1: VM migration achieves consolidated resource allocation under a low workload level. Moreover, it could ensure the QoS of cloud service users by Phase 2: the DRS procedure increases adaptively the number of active servers under a high workload level. Through experiments based on a practical use case, our proposed scheme is not only feasible from a theoretical point of view but also practical in a real cloud environment. In future work, we will demonstrate that our proposed algorithm outperforms existing approaches for energy efficient resource management through various experiments based on the implemented system in practice.

References

1. Prediction. Available: http://searchstorage.techtarget.com.au/articles/28102-Predictions-2-9-Symantec-s-Craig-Scroggie
2. R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges (2010)
3. S.A. Baset, Cloud SLAs: present and future. SIGOPS Oper. Syst. Rev. **46**(2), 57–66 (2012)
4. J. M. Myerson, *Best Practices to Develop Slas for Cloud Computing*. (IBM Corporation, New York, 2013) p. 9
5. A. Shankar U. Bellur, *Virtual Machine Placement in Computing Clouds CoRR*, abs/1011.5064 (2010)
6. M. Lin, A. Wierman, L.L.H. Andrew, E. Thereska, Dynamic right-sizing for power-proportional data centers. IEEE/ACM Trans. Netw. **21**(5), 1378–1391 (2013)
7. Z. Xiao, W. Song, Q. Chen, Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1107–1117 (2013)
8. Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, C. Pu, An analysis of performance interference effects in virtual environments. IEEE Int. Symp. Perform Anal. Syst. Softw. 200–209 (2007)
9. C.D. Patel, A.J. Shah, Cost model for planning, development and operation of a data center. Development **107**, 1–36 (2005)

10. W.-J. Kim, D.-K. Kang, S.-H. Kim, C.-H. Youn, Cost adaptive vm management for scientific workflow application in mobile cloud. Mob. Netw. Appl. **20**(3), 328–336 (2015)
11. K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, K. De Bosschere, *Performance prediction based on inherent program similarity PACT*, vol 9 (Seattle, washinton 2006), p. 114
12. Memcoder. Available: https://linux.die.net/man/1/mencoder
13. Eucalyptus. Available: http://www.eucalyptus.com
14. K. Hoste, L. Eeckhout, Microarchitecture-independent workload characterization. IEEE Micro **27**(3), 63–72 (2007)
15. A. Ali-Eldin, J. Tordsson, E. Elmroth, M. Kihl, *Workload Classification for Efficient Auto-Scaling of Cloud Resources*. (2005)
16. OpenStack. Available: http://www.openstack.org/
17. D.-K. Kang, F. Al-Hazemi, S.-H. Kim, M. Chen, L. Peng, C.-H. Youn, Adaptive VM management with two phase power consumption cost models in cloud datacenter. Mob. Netw. Appl. **21**(5), 793–805 (2016)
18. M. Chen, Y. Zhang, L. Hu, T. Taleb, Z. Sheng, Cloud-based wireless network: virtualized, reconfigurable, smart wireless network to enable 5G technologies. Mob. Netw. Appl. **20**(6), 704–712 (2015)
19. M. Chen, H. Jin, Y. Wen, V. Leung, Enabling technologies for future data center networking: a primer. IEEE Netw. **27**(4), 8–15 (2013)
20. F. Xu, F. Liu, L. Liu, H. Jin, B.B. Li, B.B. Li, iAware: making live migration of virtual machines interference-aware in the cloud. IEEE Trans. Comput. **63**(12), 3012–3025 (2014)
21. D. Gupta, L. Cherkasova, R. Gardner, A. Vahdat, Enforcing performance isolation across virtual machines in xen, Proceedings 7th ACM/IFIP/USENIX international conference middleware, pp. 342–362, (2006)
22. A. Nisar, W.K. Liao, A. Choudhary, Scaling parallel I/O performance through I/O delegate and caching system, *2008 SC—International conference for high performance computing* (Storage and Analysis, SC, Networking, 2008)
23. M. Chen, Y. Zhang, Y. Li, S. Mao, V.C.M. Leung, EMC: Emotion-aware mobile cloud computing in 5G. IEEE Netw. **29**(2), 32–38 (2015)
24. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
25. YOCTO-WATT. Available: http://www.yoctopuce.com/EN/products/usb-electrical-sensors/yocto-watt
26. G-Technology. Available: http://www.g-technology.com/products/g-drive
27. PowerWake. Available: http://manpages.ubuntu.com/manpages/utopic/man1/powerwake.1.html
28. Montage. Available: http://montage.ipac.caltech.edu/

Chapter 3
Cost Adaptive Workflow Resource Broker in Cloud

3.1 Introduction

As scientific applications become more complex, the management of resources that perform the workflow jobs has become one of the challenging issues [1, 2] Workflow scheduling is one of the key issues in the management of workflow executions. Scheduling is a process that maps and manages executions of inter-dependent tasks on distributed resources. It includes allocating suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. An appropriate scheduling can have significant impact on the performance of the system. In general, the problem of mapping tasks on distributed resources belongs to a class of problems known as NP-hard problems [3]. Even though the workflow scheduling problem can be solved by using exhaustive search methods, the time taken for generating such solution is very high. Since resource status is changing rapidly due to the competition among users, scheduling decisions must be made in the shortest time possible in Cloud environments. A number of best effort scheduling heuristics such as Min-Min [4] and HEFT [5] (Heterogeneous Earliest Finish Time) have been applied to schedule Cloud workflows. These best-effort scheduling algorithms attempt to complete execution within the shortest time possible. They neither have any provision for users to specify their Quality-of-Service (QoS) requirements nor any specific support to meet them. However, many workflow applications in both scientific and business domains require some certain assurance of QoS. For these applications, the workflow scheduling applied should be able to analyze users' QoS requirements and map workflow tasks onto suitable resources such that the workflow execution can be completed to satisfy their requirements. In addition, in order to satisfy QoS requirements due to the dynamic nature and uncertainty of Cloud, it is required for a workflow management system to provide management functions for managing workflow execution and resource management functions. Even though there are many efforts to execute workflow applications with QoS guarantee,

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_3

they provide workflow management functionality without consideration of user requirements or Service-Level Agreement (SLA), such as performance, reliability, and others. Workflow management systems in Cloud are becoming widespread for scientific applications to solve sophisticated problem such as drug discovery and high energy physics. Even though various research efforts are being made, consideration of QoS requirements is still the challenging issue.

3.2 Background and Related Works

3.2.1 Workflow Control Schemes

There were many workflow control scheme for using computing resources in efficient manner. Because workflow scheduling problem is NP-complete without some constraint [6] we need simple algorithm to adapt schemes in real system. In [5], they proposed new heuristic algorithm named HEFT for scheduling workflow tasks in heterogeneous computing environment for high performance and low cost. In this book, they calculate the downward rank which is time distance with exit node. High rank means that node is important in reducing the processing time of whole workflow, so priority is set in descent order of rank. After making the decision of priority, each node is allocated to node that needs least time consuming. This algorithm shows the good performance but it only considers the bounded number of heterogeneous processors and because of finite of resource quantity and budget, we need workflow scheduling scheme with budget constraint. In [7], they proposed new algorithm to satisfy the budget constraint by finding the best affordable assignment possible using HEFT algorithm. In some limit of budget, they schedule the workflow processing to achieve minimum execution time. They reassign resources to tasks with two approaches LOSS and GAIN until all possible reassignments would exceed the budget. There was book also about workflow model fragmentation for distributed execution [8]. For executing workflow in distributed environment, they proposed dynamic model fragmentation method include the enhanced scalability by outsourcing and the increased flexibility by designating execution site on-the-fly. These algorithms might well works in grid environment, but they not consider the feature of cloud computing's full hour billing model and unbounded amount of resources, so their algorithm is not compatible with cloud computing environment. There are also workflow scheduling schemes for cloud environment [9], but for solving complex problems in heuristic way, they use complex approaches. And also they do not consider the data transmission cost in depth. Because of data intensiveness of scientific workflow environment, algorithm needs more modification. So in this book, we simplify the scheduling problems to not consider full hour billing model so resource utilization problem is displace to resource management schemes. And we also consider the data transmission cost detail in this book.

Data-driven scientific workflows maximize the total completion time by paraltlelizing or pipelining on each task under assumptions that the input data can be divided by the sub-tasks or each stage of the task can be executed by incremental data processing methods.

The relationship between data parallel scientific workflows, two broad categories are described.

A. **Task parallelism approaches**

Single Program Multiple Data (SPMD) is a parallel processing technique where the same program (or task, in workflow terminology) is applied to many independent data items. All data items are processed in parallel and independently. Since they have no dependencies, the data parallelism pattern is applied to reduce the total completion time under parallel computing paradigm. The pattern can be classified by the methodologies of partitioning the total dataset as in [10].

- Static parallelism—The number of partitions is known in advance at design-time and it is fixed for all workflow executions.
- Dynamic parallelism—The number of partitions can only be determined at run-time to provide the portable workflow definition across multiple execution environments. In addition, the degree of partitioning an input dataset for a task is another issue to optimize.
- Adaptive parallelism—The number of partitions are automatically estimated during the execution of the workflow itself. The model indicates that the total execution time of a workflow that uses data parallelism to speed up processing of a fixed-size input dataset, is highly sensitive to the number of data partitions with respect to the amount of available resources.

B. **Pipelined execution approaches**

Since using the task parallelism pattern can reduce the execution time of one or more tasks applied in parallel to a vector of input data elements, the pipelined execution pattern, which interleaves a sequence of more than one task, is applied sequentially to a vector of input data elements. The applied approaches are introduced in [10].

- Best effort pipelines—no guarantees are provided and data is simply dropped in case of pipeline collisions.
- Blocking pipelines—a task can only be (re)started if the predecessor has completed and if its successor is idle. It guarantees the data produced by a task will not be overwritten, as the downstream tasks will be ready to process it.
- Buffered pipelines—providing buffering semantics to store the results and accumulating all intermediate results produced by a task in case the one following it in the pipeline is not yet ready to process a new input data element.
- SuperScalar Pipelines—dynamically creating additional task instances whenever they are needed to avoid collisions. It can speed up the total throughput of the workflow pipeline by adjusting the average expected throughput of each task on the workflow.

3.3 Objectives

Many research work reported various workflow tools to build simulation environments that support distributed execution. We address some problems in bio-computing using scientific workflow system, e.g. bio-computing for genome analysis, drug-finding simulation, virtual cell modeling and others. First, the existing workflow systems seldom focused on the end-to-end development life cycle of workflows, for example, design frameworks for identifying the experiment data that are ever used for computing are not provided in a real-time manner. Second, the conventional workflow systems do not support the cost-minimized VM management in Cloud. The cost-minimized VM management model in here includes cloud brokering for executing high performance tasks under cloud environments. Therefore, we have to consider a new workflow scheduling model for cloud resource management. This approach requires cost-adaptive workflow control and components to build cloud broker system. In addition, the conventional systems did not provide dynamic workflow scheduling algorithm for the resource allocation on component remote execution for performance issues. This leads to difficulty of guarantee the required QoS according to bio-computing applications, which is a mandatory prerequisite for the clouds.

Most of scientific applications fall into the interdependent task model which is called workflow application. The problems are described in the cloud resource broker for managing such workflow applications. Let assume a cloud resource broker G, represented by a three-tuple $\mathrm{G} = (\mathrm{T}, \mathrm{U}, \mathrm{R})$, to execute scientific workflow applications. The G, consists of a set of k consumers (individual scientists, science communities) $\mathrm{U} = \{\mathrm{U}_1, \mathrm{U}_2, \ldots, \mathrm{U}_\mathrm{k}\}$, a set of m resource providers $\mathrm{R} = \{\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_\mathrm{m}\}$, where the resource provider could be an in-house cluster center, private cloud, or public cloud providers. A set of n tasks $\mathrm{T} = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_\mathrm{n}\}$ are submitted to the broker by consumer or their applications, scheduled by the workflow scheduling scheme S based on the interaction between the broker and the resource providers, and finally executed by the providers. Hence, the complexity of scientific applications is mainly from migrating cloud services. Currently, cloud providers and third party cloud services offer schedule-based and rule-based mechanisms to help users automatically.

3.3.1 *Guaranteeing SLA*

A workflow application is submitted to the cloud by users, scheduled by the workflow scheduling scheme based on the interaction between the broker and the resource providers, and finally executed by the providers. So, there is no particular interaction between the users and the service providers, which are decoupled by the broker. The three roles need to express their requirements and facilitate scheduling decision to further achieve their objectives. We therefore utilize SLA that is usually defined in the community as a business agreement between two of them to create the common understanding about services, responsibilities, and others.

3.4 Proposed System Model for Cost-Adaptive Resource Management Scheme

In this section, we discuss cost-adaptive resource broker integrated workflow management system in terms of its features, architecture and functional description.

3.4.1 Assumption

We assume that a cloud broker is designed for Biocomputing application. In LCW system, there are many services/applications and different services/ applications have different resource requirements. The resources in LCW system are in the same sub-network, virtual machine (VM) in cloud environments. Different resources have different capabilities.

3.4.2 Requirement Descriptions

We raised the problems of existing workflow management systems and proposed solutions to improve the policy-adjuster-integrated workflow management system based on cloud broker system. To solve the problems we mentioned above, our proposing system should have following features:

- In-time workflow scheduling mechanism. Owing to the dynamic characteristic of distributed resources, an in-time scheduling mechanism for workflow management is needed.
- A SLA modeler. To guarantee the SLA for the workflow execution, SLA requirements from users should be received and handled in our system.
- Policy selection mechanism. To manage the policies for different users' intention, a decision scheme is needed to choose the suitable policy for the workflow scheduling based on the modeled SLA.
- Runtime prediction mechanism. Since the deadline time factor is included in SLA, we should have a mechanism to predict the future runtime of a task in our system. The time prediction will be realized through analysis of historical execution data.

- Resource monitoring. To allocate the sub-jobs of workflow to the dynamic and heterogeneous Grid resources, a resource monitoring function is required. This function is provided through cloud broker system.
- Task monitoring. To control the workflow execution sequence, a task monitoring function should be built to get the task status including submitting, waiting, executing, and done.

3.4.3 A Layered Cloud Workflow System (LCW)

Although the available resources can guarantee user's service level in the current status, however, we cannot keep the promise we made earlier all the time, since the later job executions may affect the earlier job's performance. Also the ways we guarantee current user's service level may bring a significant cost to the system. In LCW, we bring information from application layer and cloud together into the workflow system to improve the performance. Before user submits a job, we will login to our system, in which we have user policy to classify the user into groups. When the user chooses the service from the cloud service lists provided by LCW system, we can divide the jobs into different work process according to their types. The work process information and user service level information are formed in the application layer. In the cloud, the resource broker will decide how to distribute jobs into sub-jobs and how many pieces should we divide a whole job to also how many resources should the sub-jobs occupy. We do not consider the detailed sub-job dependences because they are highly related to specific applications. When all sub-jobs are processed, the cloud gather the results and present them to the user. Figure 3.1 shows the detailed module of cloud resource broker in workflow policy integrated to LCW middleware: Resource management in clouds consists of resource discovery and selection, job scheduling on the selected resource, and job monitoring and migration to replicas.

To improve QoS and guarantee reliable resource management in clouds, we propose LCW as shown in Fig. 3.2. First, more information is taken into consideration to increase the accuracy. We can improve performance through managing resources efficiently. Second, we can expect efficient distribution of the sub-jobs. Third, we can expect more efficient resource management by integrating workflow management with LCW resource management scheme. In this architecture, workflow module delegates a policy adjustor and checker to manage job processing through clouds.

The platform service specialized on bio workflow computing on top of various computing environment and core service stacks for encapsulating applications on cloud. To negotiate among workflows in a real-time manner, we describes a conceptual idea of the integrated workflow system. The middle layer includes functions which manage the interaction among different bio-workflow services, such as the next generation genome sequencing (NGS), MapChem Broker, and RVR-based

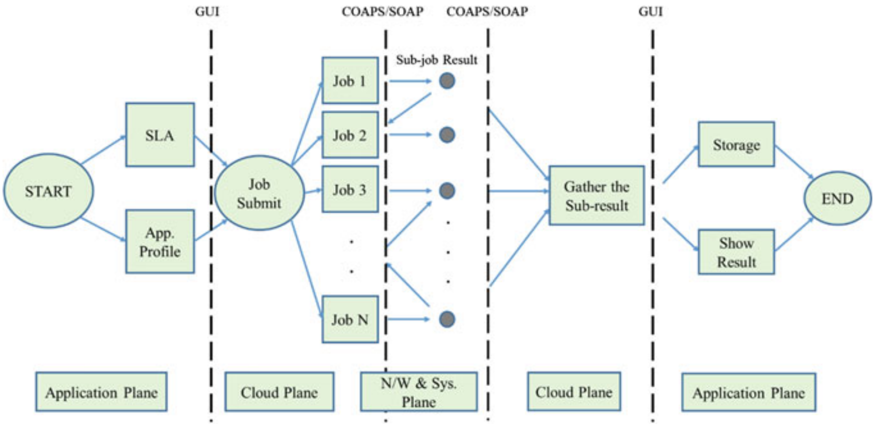


Fig. 3.1 Workflow job scheduling scheme integrated in cloud computing environment

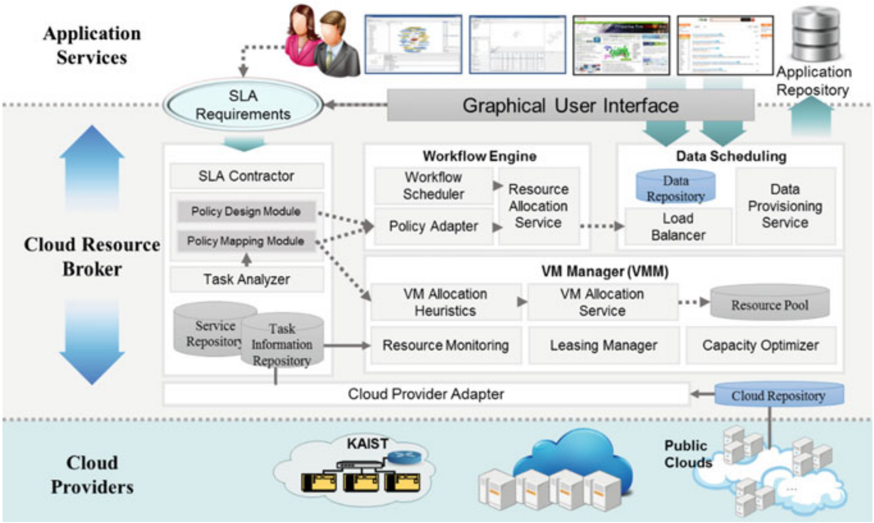


Fig. 3.2 Proposing cloud Resource broker system

BioMap Systems implemented in workflow model. The system could automate the simulation data and experiment workflow management. Such large scale data analysis workflow model needs huge size of computing and storage infrastructure for performing overall workflow tasks in an in-house method. However, it is expensive to prepare enough resources and the efficiency of resources is relatively low since all tasks do not require the same computing capacity. On the hybrid cloud model, it is possible to outsource the entire or a part of the workflow tasks into the public cloud. In this case, on-demand resource provisioning is possible whenever it is needed. To control the workflow execution flow, a task monitoring function is built to identify the task status including submission, execution, and publish.

Workflow scheduling engine Workflow scheduling manager provides a graphic user interface (GUI) for a legacy application service of an arbitrary user. We focus on job processing applications such as bag of tasks (BoT) [11] or scientific workflow applications [12] which are a kind of embarrassingly parallel job processing based on a global job queue. The applications are deployed by an administrator with additional QoS related components, especially, enabling automatically providing the required number of the resources (virtual machine (VM) typed compute servers) through workload sensing and threshold based trigger mechanism. Because the proposed system is also possible to process simple and independent task based applications, Biocomputing applications are easily expressed by a set of inter-dependent jobs—workflow.

Workflow scheduling manager is responsible for managing and executing these workflows. From the GUI tools, users compose the workflow specification with a set of available services that is provided by the broker. After parsing the user's submitted job specification, the workflow scheduler partition the entire workflow into simple tasks with its dependency map. The dependency map determines the start time of the tasks. When a kind of SLA requirements, such as deadline or budget limit, comes from policy adapter, the scheduler should consider and apply requirement on the scheduling scheme. Finally, the result of the workflow scheduling are expressed by the task to resource mapping table and it is stored in the task queue. So the resource allocator is responsible for allocation each task with resources at run-time.

VM resource manager The resource provisioning manager, which manages the logical resource, is the core of our cloud broker. We define a virtual machine pool (VMP) as a logical container which stores a set of resources with different types. The resources in the VMP represent the prepaid resources that is physically deployed in the underlying IaaS providers. So, the provisioning manager allocates or deallocates the resources in VMP by sharing among various applications. Especially, the allocation manager accepts the requests from the resource allocator in the workflow scheduling manager. The VMPM seeks the optimal VMP size by combining different VM leasing policies to minimize the cost.

Policy enactment manager Since the workflow management domain and resource provisioning domain consider different aspects, there are some conflicts between two different management systems, such as service type of each node, execution environment, user requirements, etc. In cloud, the complexity and cost of resource management for QoS guarantee significantly increases as the scale of the cloud increases. For the reliable management of leasing resources, there have been some approaches with attemptation to solve this problem. One of the most promising resource management schemes is the policy-based resource management system that is suitable for complex management environments (Fig. 3.3).

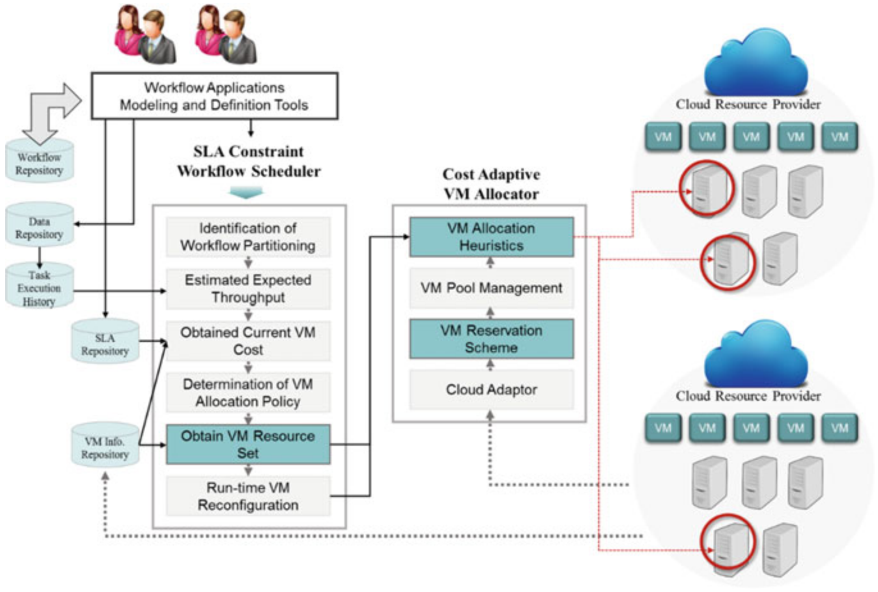


Fig. 3.3 Workflow computing integrated cloud resource broker

3.5 Proposed Cost Adaptive Workflow Scheduling Scheme

In our broker system, we assume there are three roles: the cloud resource broker, users' workflow applications, and the cloud resource providers. The users submit their applications/workflows/data to the cloud resource broker; the broker will buy or release resources from/to the cloud when needed; and then the broker will allocate an appropriate VM instance to execute each task of the workflows.

Users' applications are in users' computers. The application and relative data need to be firstly transferred to the cloud resources before execution. A workflow consists of applications with constrained precedence. Each application is a task of the workflow and each task will be mapped to some cloud resources for execution. Each task has different performance on different types of VM instances. Workflows may have different levels of importance and urgency; therefore users can specify different deadlines for workflows. Users can submit workflows at any time and the cloud resource broker has no knowledge about the incoming workload in advance.

The Cloud resources can be categorized into several types according to their sizes, such as small, medium, large, and xlarge. Each type of VM will offer different processing power. VM instances are charged in terms of per-hour usage. Less-than-one-hour consumption is rounded up to one hour.

3.5.1 Workflow Resource Allocation Optimization Problem

In order to obtain the optimal resource allocation state for a given workflow, we should determine the ratio of the resource allocation for each task to minimize the completion time of the overall workflow. The following conditions should be satisfied.

- For a replicated task S_k, there exists an optimal ratio of the resource allocation of the replicated tasks, which minimizes the completion time in the tasks. Also, for a parallelized task S_k, there exists an optimal ratio of the resource allocation of the parallelized tasks, which minimizes the completion time in the tasks.
- For an arbitrary pipelined and constrained workflow, say W (Y, Φ, D), where Y, …, there exists an optimal ratio of the resource allocation which has both parallelization and replication, and minimizes the total completion time.

Since the pipeline workflow can be divided by the each tasks and can allocate the deadline of each task, on the condition that the estimated completion time is known, we can obtain the optimal solution of the each task. The following equations should be ordered, mentioned and explained.

$$\begin{array}{ll} minimize & T[w] \\ subject\,to & T[w] = max_{l \in L, m \in M}\left\{T\left[w_{l,m}(t)\right]\right\} \\ & T[w] \leq T_{deadline} \end{array} \tag{3.1}$$

3.5.2 Obtaining Expected Throughput Based on Estimated Completion Time

We obtain the expected throughput (ETH) for each resource type i (ETH_i) based on estimated completion time. The completion time of each task is different from the input parameters of the task. So, we estimate it based on historical information of each task with the reference instance.

We exploit the completion time estimation for different schemes under the assumption that new task does not has any historical information, and the execution coefficient based on historical information.

A. **Completion time estimation of new task**

For each submitted workflow, the broker will complete the workflow within users' specified deadlines. First it assigns a suitable type of VM to each task; then, the broker assigns a suitable instance of the selected VM type to the task. In short, the broker needs to make a mapping relationship between each task and some VM instance. For this mapping decision, the broker will adopt different policies that are implemented with different algorithms within the broker. Assume the cloud workflow broker can always estimate the task processing times on different types of VMs. The cloud workflow broker maintains the appropriate volume of its VM pool by auto-scaling, so that it can save the monetary cost of buying cloud resources.

We apply workflow scheduling mechanism and resource auto-scaling scheme to finish all users' requests within users' specified deadlines in order to minimize the total cost for purchasing the processing resources in the cloud. The workflow scheduling mechanism instructs the broker to firstly map each task to be executed to some appropriate VM type, and then schedule the task to a VM instance of that type at the particular time slot. The resource auto-scaling scheme enables the broker to keep a certain number of VM instances of each type in the resource pool, which have right enough processing power to meet the dynamically changing workload from users and also can avoid the broker's unnecessary purchase for abundant VM instance.

Algorithm 1 Completion time estimator

```
INPUT: A set of tasks and their data set
OUTPUT: Prediction of the completion time for candidate resource set.

BEGIN:
  while all tasks in the workflow do
    while all resource types in the cloud resource broker do
      Calculate the estimated execution time for each task on resource type $R_i$ based on Eq.
      (2) and (4)
      Create estimated completion table
    end while
  end while
END
```

At some time t, the workload for a VM type R_i in the broker's current VM pool are the tasks that are being executed and waiting to be executed on this specific VM type; it is denoted as Wl_t^i. The broker needs to maintain a proper number of instances of each VM type in the VM pool. The number of all the instances of type R_i in the VM pool at time t is denoted as $N_t^{R_i}$. An exact performance estimation for each task on each VM type is critical in the later scheduling decisions. The broker can estimate the performance of a task on different types of VM, by using the following performance estimation technique.

Also, we can estimate each tasks' execution time. The estimated execution time of a task on a VM consists of two parts of time, i.e., the data transfer time and the application running time. For running task S_k on VM type R_i, the estimated execution time $T^{ee}(S_k, R_i)$ can be obtained by Eq. (3.2).

$$\exists i, T^{ee}(S_k, R_i) = T_{transfer} + T_{execution} = \Sigma \frac{I_i^n}{r_{n_i}} + \frac{\omega_i}{r_{c_i}} \tag{3.2}$$

B. **Completion time estimation of new task**

For an arbitrary task, S_k, we define a parameter set $P = \{p_{k,0}, p_{k,1}, \ldots, p_{k,q}, \ldots, p_{k,Q}\}$ where the parameter $p_{k,q}$ is a factor that affects the execution time of the task selected by the workflow manager. Now, we define the execution coefficient for an arbitrary parameter $p_{k,q}$ as $\alpha_{k,q}$.

$$\alpha_{k,q} = \frac{\tilde{T}^{ee}(S_k) - T^{ee}(S_k)}{\tilde{p}_{k,q} - p_{k,q}} = \frac{\Delta T^{ee}(S_k)}{\Delta p_{k,q}} \tag{3.3}$$

We assume that the execution parameter coefficient is obtained by historical information. So, we can calculate the estimated completion time of the task D with an arbitrary parameter set P as Eq. (3.4).

$$\exists i, T^{ee}(S_k, R_i) = \prod\nolimits_{l \in Q} \left(\alpha_{k,q} \cdot p_{k,q}\right) \tilde{T}^{ee}(S_k, R_i) \tag{3.4}$$

The detailed procedure of the completion time estimation is shown in Algorithm 1. Finally, based on Eq. (3.3) and Eq. (3.4), we obtain the expected throughput for each VM type for the task as

$$\exists i, ETH(S_k, R_i) = \frac{T^{ee}(S_k, R_i)}{Unit\_Time} \tag{3.5}$$

where *Unit_Time* represents the time duration in terms of sec, min, and hour, or the deadline which is enforced by user's SLA. Since the VM cost is calculated per hour, The expected cost during the time interval T, $EC(R_i, T)$ is shown in Eq. (3.6).

$$EC(R_i(t), T) = C(R_i(t)) \left\lceil \frac{T}{T_{hour}} \right\rceil \tag{3.6}$$

3.6 Proposed Marginal Cost Based Resource Provisioning Scheme

In this section, we describe the functionalities of the VM provisioning manager (VMPM) and its policies in terms of reducing the resource leasing cost. Before further describing those, we first define the fundamental terminologies used in this chapter. A vector $A = \{A_1, A_2, \ldots, A_a, \ldots, A_A\}$ denotes a set of cloud application services (ASs) deployed in the broker. Each CASE has an individual application logic with the service platform. The CASE adds the computing resources by requesting to the VMPM and removes the resources when they are useless any more. The arrival density of the resource requests and the usage duration is perfectly dependent on each CASE. So, the VMPM cannot control those parameters.

A vector $R = \{R_1, R_2, \ldots, R_r, \ldots, R_R\}$ denotes a set of resource capacity types which are available in the broker. In the broker, since such the resources are provided by the underlying IaaS cloud providers, they can be indexed by the logical resource pool. We use only ready-made types of the VMs such as small, medium, large, xlarge. So the application services request the resources by specifying such the resource-type.

A vector $L = \{L_h(R), L_d(R), L_m(R), L_{6m}(R), L_y(R)\}$, where h, d, m, $6m$, y refer to the payment plans such as a hour, a day, a month, 6-month, and a year respectively. We assume that the unit price is cheaper when the duration is longer e.g., $L_h(V) \geq L_d(V) \geq L_m(V) \geq L_{6m}(V) \geq L_y(V)$. Without loss of generality, we simplify the leasing types as two types - on-demand VM as OVM $(L_h(V))$, hourly paid and reserved VM as RVM (L_m), monthly paid since most of the public cloud providers adapts those leasing policies. So, in the broker level, the resource allocation state is expressed by such three entities. On the following section, we describe the operation procedure for resource allocation on the VMPM in detail.

3.6.1 VM Resource Allocation Procedure

When deploying each application services, it has the VM configuration policy that denote the minimum and maximum size of each set of V_j as $A_{a,r}^{min}$ and $A_{a,r}^{max}$. So, the initial number of the VMP is

$$\forall r, N_r(0) = \sum_{a \in A} A_{a,r}^{min} \tag{3.7}$$

and the following capacity constraints should be satisfied in the VMP:

$$\forall r, \sum_{a \in A} A_{a,r}^{min} \leq N_r(t) \leq \sum_{a \in A} A_{a,r}^{max} \tag{3.8}$$

In addition, the $N_r(t)$ is divided by two set of payment plans such as.

$$\forall r, N_r(t) = N_r^{(m)}(t) + N_r^{(h)}(t) \tag{3.9}$$

where $N_r^{(m)}(t)$ and $N_r^{(h)}(t)$ refer to the number of reserved VMs (RVMs) on-demand VMs (OVMs), respectively.

On the view of the VMPM, it has set of incoming events and outgoing events as follows,

- $aReq(a, r)$ is a request event for resource type r from the application service a. The response of the event is $aReqs(a, r, index)$, where the index is the global identification of the allocated resource.
- $dReq\_req(a, r, index)$ is a remove event for the indexed resource from the application service a. The response of the event is $dRes(true)$, where true is the success of the remove.
- $lReq(r)$ is a resource leasing request event for resource type r to the resource provider. The response of the event is $lRes(r, index)$, where the index is the identification of the leasing resource.
- $rReq(r, index)$ is a release request event for indexed resource to the resource provider. The response of the event is $rRes(true)$.

The incoming events are the requests to allocate or deallocate the VMs to/from an arbitrary application service. On the other hand, the outgoing events are the requests process to lease or release the VMs from/to the cloud resource provider. Those operations are responsible for the Allocation Manager (AM) and Leasing Manager (LM) in the VMPM. We index each VM as $index = [id, state, R, A]$ where the R, A are the resource type, and index of application service respectively and the $state$ refers to the 'allocated' or 'available' (deallocation) of the resource (on the view of general instance states, the allocation state is the 'running' and the available state means the 'pending' state of the instance. Also, the index is created when the VM is leased and deleted when the VM is released.

Overall operation procedure of the VMPM is described in Algorithm 2. Let the reservation capacity (number of RVMs) for the resource type r be i_r^*. Usually, the size will be the sum of the minimum number of resources for all applications. Prior to starting the service, the VMPM determines the i_r^* for all resource types. After determining the reservation capacity, the VMPM leases a set of RVMs as much as minimum required VMs e.g., $\sum_{i \in I} A_{i,j}^{min}$.

When an allocation request comes, VMPM allocate the longest available (latest leasing (RVM to request. If there is no available RVM exist and if we can affordable to lease the RVM (current number of RVMs are less than reservation capacity), then, VMPM lease a RVM from resource provider and allocate the resource to the request.

Algorithm 2 VMPM resource allocation policy

```
$\forall j$
Determine initial number of RVMs as $i_r^* = \sum_{i \in I} A_{i,j}^{min}$
while VMPM is operated do
  if An allocation event $aReq(a, r)$ comes then
    if Available RVMs exist then
      Allocate the longest available RVM to $aRes(a, r, index)$
    else
      if $N_r^{(m)}(t) < i_r^*$ then
        Lease a RVM $(R_r)$ from resource provider
        Allocate the request to $R_r$ and send $aRes(a, r, index)$
      else if
        Lease a OVM $(R_r)$ from resource provider
        Allocate the request to $R_r$ and send $aRes(a, r, index)$
      end if
    end if
  end if
  Change the stat as 'allocated' for the indexed RVM
  else if
    if The VM is RVM then
      Change the stat as 'available'
    else if The VM is OVM then
      Release the VM and delete the index
    end if
  end if
end while
```

However, if the capacity is full, then, VMPM lease a OVM from resource provider and allocate the resource to the request. Once, the resource is allocated, the status of the resource should be set to the 'allocated'. On the other hand, if the deallocation event comes and the resource is RVM, the status of the resource set to the 'available'. Meanwhile, if the resource is OVM, the VMPM release the resource and delete the index.

As shown in the operation procedure, it is the simplest and naive procedure of the VMPM. Especially, adjustment of the capacity of the RVM does not apply since it needs more complicated optimization process. In the following section, we describe the cost-minimized capacity control policy based on Markov Decision Process.

3.6.2 Marginal Cost Based Adaptive Resource Reservation Scheme

We propose the adaptive resource reservation scheme (ARRS) which is based on marginal cost (MC). At first, we obtain the total cost which is the summation of the costs for the RVMs and the OVMs during $(0, T)$. Supposing $N_r(t)$ is the number of r-type resource which is leased from providers at time t, the total cost is expressed by

Algorithm 3 Adaptive resource reservation scheme (ARRS)

```
$\forall j$
Set cost and periods for the RVM and OVM, $\xi_r^{(m)}, \xi_r^{(h)}, T_r^{(m)}, T_r^{(h)}$
Initialized total cost parameters for RVM and OVM, $C_r^{(m)} = 0,\ C_r^{(h)}(0) = 0$
Set ARRS monitoring period, $T$
Set initial RVM as much as minimal requirement size of applications
while ARRS is operated do
  if current time $t$ equals to $T$ then
    Get current total cost $C_r(t) = C_r^{(m)}(t) + C_r^{(h)}(t)$ from cost analyzer
    Calculate Marginal Cost, $MC = \frac{\Delta C_r^{(h)}}{\Delta C_r^{(m)}}$
    if MC is greater than 1 then
      Lease RVM as much as $\lfloor MC \rfloor$
    else
      Release a RVM
    end if
    Set next decision point $T = t + T$
  end if
end while
```

$$C_r(t) = C_r^{(m)}(t) + C_r^{(h)}(t) = T\left[\frac{\xi_r^{(m)} \cdot E\left(N_j^{(m)}\right)}{T_r^{(m)}} + \frac{\xi_r^{(h)} \cdot E\left(N_r^{(r)}\right)}{T_r^{(h)}}\right] \tag{3.10}$$

where $T_r^{(m)}$ and $T_r^{(h)}$ are the unit pricing time and $\xi_r^{(m)}$ and $\xi_r^{(h)}$ are the pricing time and unit price of RVM and OVM types, respectively. Also, E(•)s represent the average terms of the inner values. On the other hand, the marginal cost of the k-type VMs, $MC_k(t)$ during $(0, T)$ is defined by

$$MC_k(t) = \frac{\Delta C_r^{(h)}}{\Delta C_r^{(m)}} = \frac{\left(C_r^{(h)}\right)_{t=t} - \left(C_r^{(h)}\right)_{t=t-T}}{\left(C_r^{(m)}\right)_{t=t} - \left(C_r^{(m)}\right)_{t=t-T}} \tag{3.11}$$

where the period for calculating the marginal cost is generally smaller than the period of the RVM, e.g., $T < T^m$. On the Algorithm 3, the ARRS is working within T^m and adjust the number of RVMs by leasing or releasing those based on the value of the marginal cost.

3.6.3 Adaptive Resource Allocation Heuristics

In this section, we propose the novel approaches called Adaptive Recycle, Replacement, and Reduce (A3R) algorithms for an efficient management of VM instances in the resource pool in order to solve the dissipation caused by unnecessary occupation of resources in the previous researches. We demonstrate that the saving of resource operation cost is achieved efficiently by using our simple methods.

A. **Resource recycling policy**

The procedure of Recycle scheme for an efficient VM allocation is shown in Fig. 3.4. Several cloud service providers such as Amazon and Google propose the several resource allocation policies to cloud service users. In particular, the VM instance is allocated to the cloud service users based on the fixed unit time basis for usage of resources. In other words, VM instances are provided to cloud service users based on a coarse-grained unit usage time of resources. For example, the usage unit time of VM instance is 1 h, then users should pay VM instances by the hour (1 h, 2 h, …, etc.) and the users occupy the VM instances until the end of the usage unit time in regardless of the completion of the requests. If the required job in the request from the user needs low computing capacity and short processing time, then the dissipation of unnecessary resource occupation can be occurred.

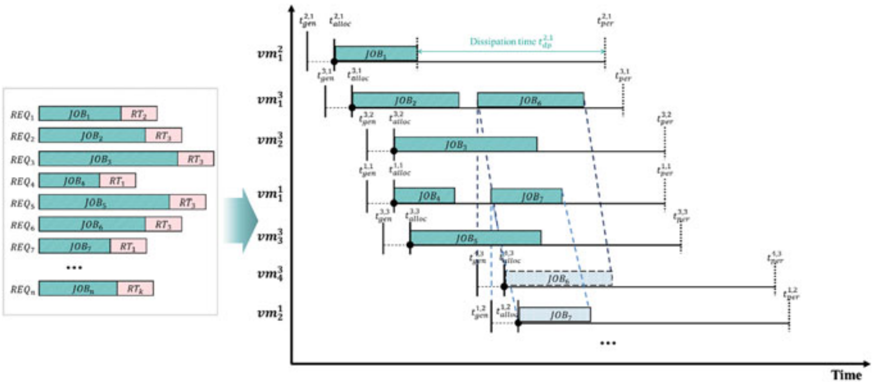

REQ_1 JOB_1 RT_2
REQ_2 JOB_2 RT_3
REQ_3 JOB_3 RT_3
REQ_4 JOB_4 RT_1
REQ_5 JOB_5 RT_3
REQ_6 JOB_6 RT_3
REQ_7 JOB_7 RT_1
•••
REQ_n JOB_n RT_k
vm_1^2
vm_1^3
vm_2^3
vm_1^1
vm_3^3
vm_4^3
vm_2^1
$t_{gen}^{2,1}$ $t_{alloc}^{2,1}$ $t_{per}^{2,1}$
Dissipation time $t_{dp}^{2,1}$
$t_{gen}^{3,1}$ $t_{alloc}^{3,1}$ $t_{per}^{3,1}$
$t_{gen}^{3,2}$ $t_{alloc}^{3,2}$ $t_{per}^{3,2}$
$t_{gen}^{1,1}$ $t_{alloc}^{1,1}$ $t_{per}^{1,1}$
$t_{gen}^{3,3}$ $t_{alloc}^{3,3}$ $t_{per}^{3,3}$
$t_{gen}^{4,3}$ $t_{alloc}^{4,3}$ $t_{per}^{4,3}$
$t_{gen}^{1,2}$ $t_{alloc}^{1,2}$ $t_{per}^{1,2}$
JOB_1
JOB_2
JOB_6
JOB_3
JOB_4
JOB_7
JOB_5
JOB_6
JOB_7
•••
Time

Fig. 3.4 The procedure of recycle scheme for VM instances

However, in traditional studies about virtual resource management, the VM instance allocation scheme is only considered as a mapping function of single user's request to single VM instance. Hence in proposed Recycle scheme, the remaining time of VM instance can be used for processing of requests from other users in contrast to previous VM management schemes in which the one certain VM instance is dedicated to only one user. Thus, in previous traditional schemes, in order to support n requests from user, n VM instances are required. That is, in regardless of the total size of requests, the same amount of virtualized resources are needed according to the number of requests. This is a quite unreasonable dissipation. In our proposed scheme, we can reduce the number of excessive allocation of VM instances by making the best use of the pre-allocated VM instances very simply.

Newly the inserted job can be processed on the pre-owned VM instances which have enough remaining time to complete the inserted job until their next end of the usage unit time. Therefore, our proposed scheme is effective when the size of requests tends to be small piece. If the group of existing resources has the same resource type with the required resource type of request, then the request can be allocated to that resource of the group. The remaining time of the certain VM instance is as follows,

$$t^{n,k}_{remain} = t^{n}_{u_{unit}} - \left(t_{proc}(Job, vm^{n}) / t^{n}_{u_{unit}} + t^{n}_{gen} \right) \tag{3.12}$$

where $t^{n}_{u_{unit}}$ is the unit time for usage of VM instance n, $t_{proc}(Job, vm^{n})$ is the processing time to complete the inserted job on the VM instance n, and t^{n}_{gen} is generation time for VM instance n. As the data size of job in the request from user is decreased, the remaining time. $t^{n,k}_{remain}$ of k-th VM instance in the resource type n, is increased, then the level of the dissipation is also increased. In addition, this means that the availability of the additional job processing is also increased. The usage time of VM instance n is as follows,

$$t^{n,k}_{usage} = t_{proc}(Job, vm^{n}) + t_{proc}(Job, vm^{n}) mod \left(t^{n}_{u_{unit}} \right) \tag{3.13}$$

Therefore, the dissipation time by unnecessary VM occupation is drawn from the difference between the processing time of job and the unit basis time for usage of resource. In Recycle scheme, since the dissipation time of resource occupation can be used to process other requests, we can not only avoid the waste of resource but also decrease the number of required VM instances efficiently and simply. In addition, we can reduce the generation delay of new VM instances for processing the inserted job by using our scheme. The processing time of the request is calculated as follows,

$$t^{n,k}_{gen\_delay} = t^{n,k}_{alloc} - t^{n}_{gen} \tag{3.14}$$

$$t_{proc}\left(REQ_{S_i \in I}, vm^n_k\right) = \sum\nolimits_{i \in I} t_{proc}\left(REQ_i, vm^n_k\right) \tag{3.15}$$

$$t_{proc}\left(REQ_{S_i \in I}, vm^n_{i_k}\right) = \sum\nolimits_{i \in I} t_{proc}\left(REQ_i, vm^n_{i_k}\right) + N(I)t^n_{gen\_delay} \tag{3.16}$$

Equation (3.15) represents the total processing time of the set I of requests in the kth VM instance of resource type n and Eq. (3.16) represents the total processing time of the set I of requests in multiple VM instances of resource type n. By above formulas, we show that our proposed scheme can reduce the generation delay of the traditional schemes.

B. **Resource replacement policy**

Figure 3.5 shows the procedure of our proposed scheme called Replacement for coping with the excessive requests in multiple resource type temporarily. As shown the figure, we assume that there are three prepared resource types such as RT_1, RT_2, RT_3 in order to provide suitable resources to users according to their request types $(RT_1 < RT_2 < RT_3)$. If the average number of usage of RT_1, RT_2, RT_3 is 5, 4, and 6, so the number of reserved VM is also defined 5, 4, and 6, respectively. But at the certain epoch, for example, if the transient usage of RT_1, RT_2, RT_3 is 6, 5, and 2, then in the cases of RT_1, RT_2, there is 1 excessive request, respectively. That is, the two idle VM instances of type RT3 can be supported to their requests instead of RT_1, RT_2 resource types.

To enable this, we should know the expected number of arrived requests during from the epoch of VM instance insourcing to returning. If the number of arrival during that period will be more than the remaining capacity, then the VM instances can't be aided for other resource groups. Otherwise, we can provide the remaining VM instances to other resource groups to support newly inserted requests without additional on-demand VM instance generation. If we use our Replacement scheme, then we can increase the utilization of the idle VM instances in certain resource type and avoid the cost caused by the additional resource allocation. The cost for additional VM instance allocation is calculated as follows,

$$\sum\nolimits_{i=1}^{n} \max a\left[N^{vm^i}_{required(t)} - N^{vm^i}_{remain(t)}, 0\right] cost\left(vm^i\right) \tag{3.17}$$

In order to enable providing the remaining VM instances in the certain resource type to other resource groups, we have to know the expected number of arrival during the period of lease. To predict the arrival number of requests during the certain period, we adopt the logistic regression model that is suitable for the filed in which the linear regression model is not reasonable.

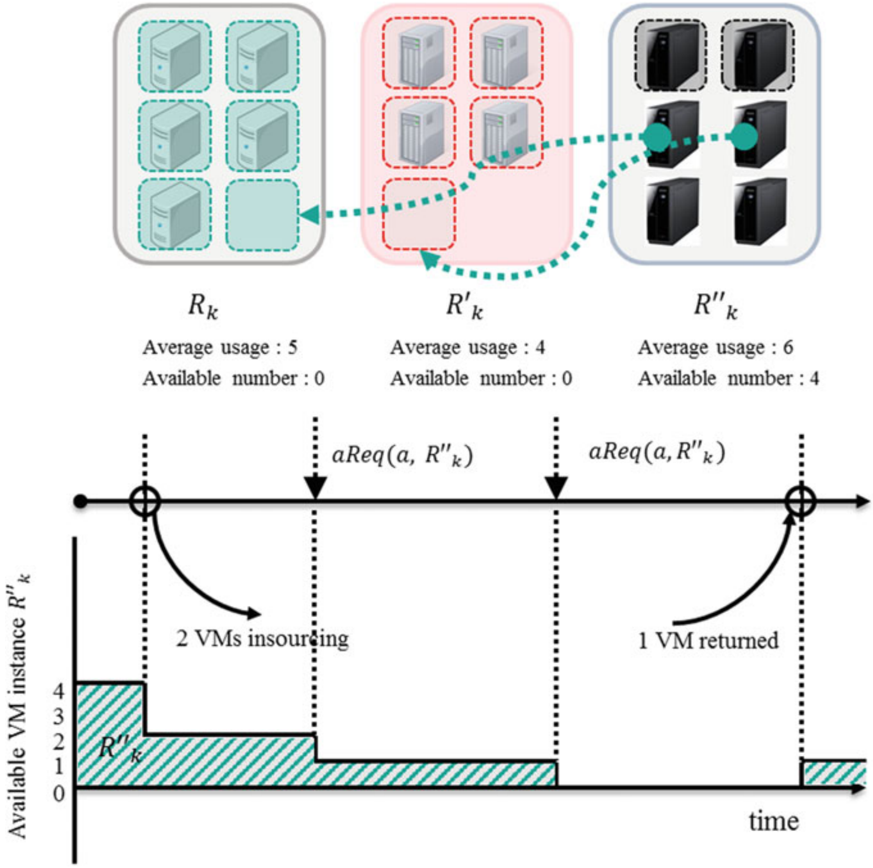


Fig. 3.5 The procedure of replacement scheme for VM instances

C. **Resource reducing policy**

Figure 3.6 represents the procedure of the Reconfiguration scheme. There are two kinds of VM instance in our environment; on-demand VM and reserved VM. Our objective is minimizing the number of on-demand VM but maximizing the utilization of the reserved VMs with acceptable QoS assurance. In Reconfiguration scheme, the ongoing processing of request from user in the on-demand VM instance can be migrated to the reserved VM if the status of the reserved VM is idle. We assume that the request consists of several tasks and they are processed in sequential order. As mentioned above, the VM allocation is provided to users based on the fixed unit time for usage of resource. Therefore, in Reconfiguration scheme, we migrate the ongoing request from the on-demand VM to reserved VM before at the end of the period for usage of resource. In addition, if the completed portion of ongoing task is significantly large, and the amount of remaining tasks is small, then

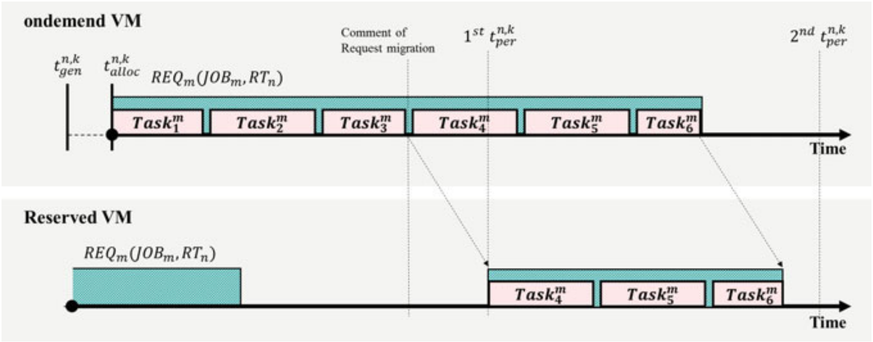


Fig. 3.6 The procedure of reactivation scheme for VM instances

we rather complete the processing of request on the on-demand VM continuously than on the reserved VM. Moreover, we should consider the migration overhead between the on-demand VM and reserved VM. The migration overhead might not only cause the undesirable additional delay to the processing of the request but also increase the cost.

$$\begin{aligned} t_{proc}\left(task_{(1,h)}^{m}, vm_{k}^{n}\right) C_{on}\left(vm_{k}^{n}\right) > t_{mg}\left(task_{(h+1,j)}^{m}, vm_{k}^{n}\right) C_{on}\left(vm_{k}^{n}\right) \\ + \left(t_{mg}\left(task_{(h+1,j)}^{m}, vm_{l}^{n}\right) + t_{proc}\left(task_{(h+1,j)}^{m}, vm_{l}^{n}\right)\right) C_{rv}\left(vm_{l}^{n}\right) \end{aligned} \tag{3.18}$$

That is, if the cost of the processing on on-demand VM is more expensive than the cost of sum of processing on reserved VM and the migration from the on-demand to reserved VM, then the Reconfiguration is suitable for the resource operation cost saving. In addition, we should consider the reprocessing of the interrupted ongoing task by VM migration.

3.7 Experiment and Results

3.7.1 Evaluation Environments

In this section, we shall describe our experiment platform and scenarios. We evaluate the proposed schemes both simulation tool simjava package [13] and experimental cloud testbed which has two different availability zones. In addition, we choose the ChemApp service [14] which is an integrated Internet-based application for collaborative pharmaceutical research. The services provided by ChemApp can be used individually, or be composed into various complex workflows in according with user's various needs. The computing intensive workload of such workflows is time-consuming. The Chem service is a perspective environment for QSAR (Quantitative Structure Activity Relationship). The perspectives it provides include drawing chemical compounds, QSAR prediction, and plotting 2D, 3D. On the testbed, the users submit the workflow jobs to the cloud resource broker with workflow XML description. We use total 32 VMs on each availability zone.

Figure 3.7 shows an example overall chemical service process. Information of customized chemical compounds is inserted into ChemApp Service System. The system can load QSAR values from Chemical DB and show the list to the user. It can also calculate 60 discriptors and predict QSAR results by using regression model; the whole process is divided into sub-tasks by the workflow service. The VM instances are generated from the Cloud infrastructure and each sub-task will be assigned to each VM instance. Finally, the results are reported to the user.

All service in ChemApp application are located in Service repository, so we can instantly reference or using the services. All services are managed by workflow for processing the jobs in regular sequence. Above figure is workflow scenario of Map service to Chem service. Local data or data from database is supplied to map service. Map service generates and find out the related chemical map and list for drug repositioning. From list user can select chemical for working in Chem Service which provide QSAR analysis and regression service. QSAR analysis needs many computing resource for calculating molecule characteristic descriptors. So it might need scheduling algorithm and distribution computing for better SLA. So we use cloud workflow for guarantee the SLA in processing environment with scheduling (load balancing), resource broker and distribution computing. After QSAR analysis, we can get the table of descriptors to molecule and its values. From value regression service generate equation of relationship between descriptors. There is three method of regression. From plotted the data and equation we can see the similarity between data and equation. These services sequences are preset by user with no programming knowledge and automatically progress with workflow manager.

Figure 3.8 describes the whole experimental testbed architecture and working procedures of the MCCS. Firstly, users submit their requests to the Flex based

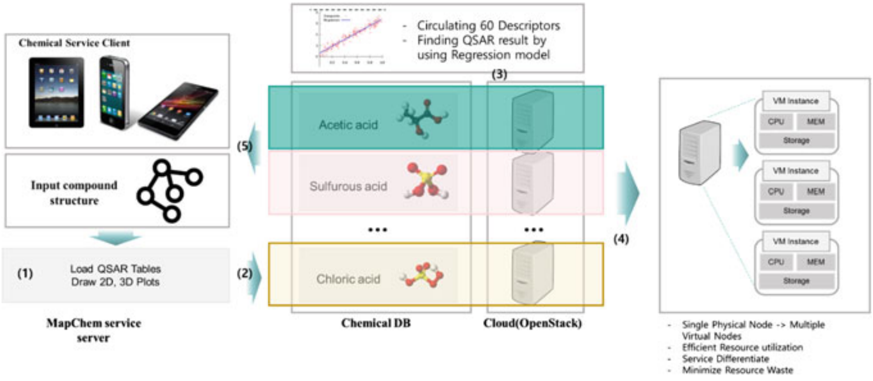


Fig. 3.7 Overall chemical service process

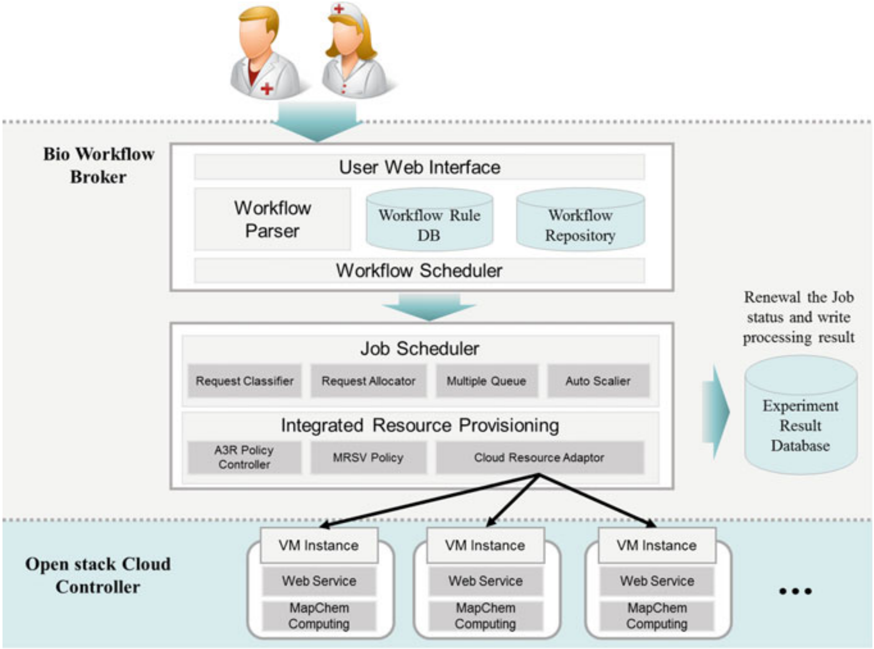


Fig. 3.8 Experimental testbed configuration

workflow designer interface. After then the workflow management module transfers the submitted workflow model as a XML description to the below web service module. The web service module using RESTFUL-based structure generates sub-xml files (that is the sub-tasks) from the whole XML description. The information of each sub-task specification is stored in the connected database system. If the web service module receives a new task that has not appeared before, then it inserts the new sub-task information into the database system. By using the policy-based adaptive workflow scheduling scheme, the whole workflow are divided into sub-tasks and then all the tasks are inserted into the job queue. Since the job queue aggregates sub-tasks from the above web service module it will then adapts the resource provisioning and auto-scaling scheme we proposed to optimize the assignment of cloud infrastructure resources and also enhance the performance of our proposed scheme.

To establish the OpenStack environment, we use five Cloud server nodes (one node as the Cloud Controller node, four nodes as the Nova computing nodes) as shown Fig. 3.9. The detailed specifications of OpenStack environment in our laboratory are as follows, Cloud controller node includes Nova module to manage the network, volume service, scheduling algorithm, image service and VM instances. In general, the role of cloud controller node is just managing the operations between Nova computing nodes, so it is not necessary to require the machine, which has good performance compared to Nova computing node.

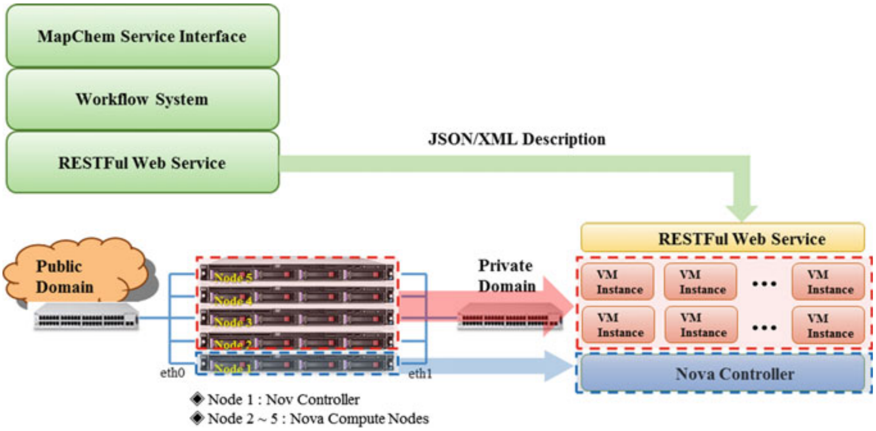


Fig. 3.9 VM configuration for experimental testbed system

Actual works are processed on Nova computing nodes and their results are reported to the Nova controller node.

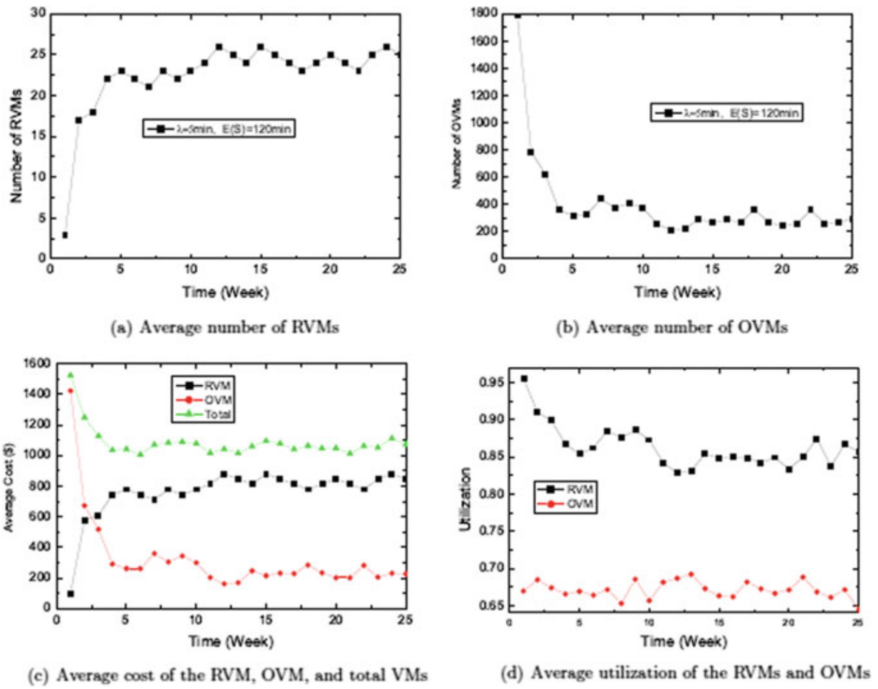
3.7.2 Evaluation of the Proposed ARRS

At first, we evaluate the characteristics of the proposed ARRS. We use the billing periods and the cost of the RVM and the OVM as the GoGrid public cloud provider [15] as shown in Table refexp1-param. Also, we set the average VM demand arrival and the average VM allocation duration as Exp(5 min) and Exp(120)min, respectively, the ARRS interval as one week, and the initial RVM as 3. In addition, the performance metrics are defined as follows:

- **The number of RVMs and OVMs**—The number of RVMs and OVMs which are leased from the cloud providers during the ARRS interval *T*.
- **RVM, OVM, and Total Cost**—The cost of VMs which is calculated during the ARRS interval.
- **Average utilization of the RVM and OVM**—Average utilization is defined by the ratio of the allocation time of the VM during the total VM leasing time.

Figure 3.10a, b show the number of the RVMs and OVMs for each period of the ARRS interval. When increasing the RVMs more than 20, the number of OVMs are decreased to average 300.

In the 5 weeks after, the number of the VMs are stable compare to the initial stats. On the stable point, the total cost of the VMs are adaptively minimized around the cost 1000$ from the initially 1500$ as shown in Fig. 3.10c. On the other hand, the average utilization of the RVM is stable on the around 0.85 as shown in Fig. 3.10d. However, the average utilization of the OVM is not affected by the



(a) Average number of RVMs

(b) Average number of OVMs

(c) Average cost of the RVM, OVM, and total VMs

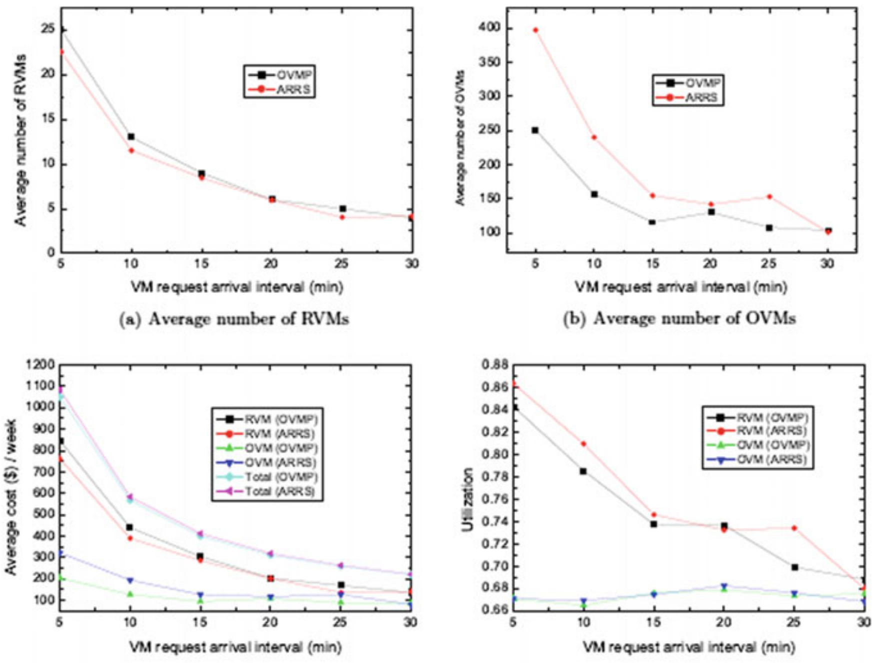
(d) Average utilization of the RVMs and OVMs

Fig. 3.10 The performance results of the ARRS

change of the number of OVMs. It cause the characteristics of the full our billing model of the OVM. Because the OVM is disposable VM, the utilization is only affect the duration of the allocation.

Secondly, we evaluate the comparison of the proposed ARRS with the OVMP algorithm which aforementioned optimal VM provisioning scheme [16]. We use the same parameters de- scribed in Table refexp1-param except different average VM demand arrivals as Exp(5,10,15,20,25,30) min, respectively.

Figure 3.11a, b show the number of the RVMs and OVMs for each period of the ARRS interval. When increasing the RVMs more than 20, the number of OVMs are decreased to average 300. In the 5 weeks after, the number of the VMs are stable compare to the initial stats. On the stable point, the total cost of the VMs are adaptively minimized around the cost 1000$ from the initially 1500$ as shown in Fig. 3.11c. On the other hand, the average utilization of the RVM is stable on the around 0.85 as shown in Fig. 3.11d. However, the average utilization of the OVM is not affected by the change of the number of OVMs. It cause the characteristics of the full our billing model of the OVM. Because the OVM is disposable VM, the utilization is only affect the duration of the allocation.



(a) Average number of RVMs

(b) Average number of OVMs

(c) Average cost of the RVM, OVM, and total VMs

(d) Average utilization of the RVMs and OVMs

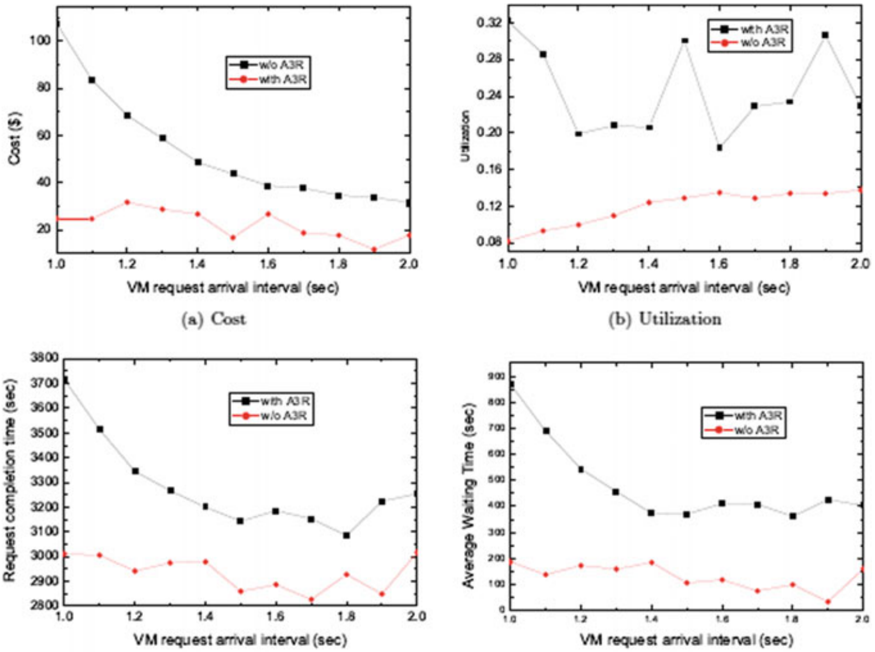
Fig. 3.11 Comparison of ARRS and OVMP algorithms

3.7.3 Evaluation of the Proposed A3R Policies

We choose the ChemApp service [14] which is a kinds of high computing bio-calculation program as an application service in order to evaluate the performance of our proposed A3R schemes compared to the conventional scheme which does not apply those heuristics. In addition, we also evaluate the average completion time and the average waiting time of the application.

Firstly, Fig. 3.12a shows the resource allocation cost of with A3R and without A3R. As shown in figure, the cost without A3R is higher than the case with A3R on all range of the VM request arrival interval. Especially, the demand rate is larger, the accumulated cost without A3R is increase rapidly. However, when the case with A3R, the total cost is maintained around the 20 30 regardless of the input rate. On the Fig. 3.12b, it shows the utilization of the both cases. In case with A3R, the utilization of the VMs are always higher than the case without A3R. It means that the A3R increase the utilization of both RVM and OVM.

On the other hand, the Fig. 3.12c shows the average completion time of the application. As shown in the plot, the completion time with A3R is a little smaller than the case without A3R. It causes the start-up time of the VMs, e.g., in the case with A3R, the VMs are often reused, so, the start-up time is eliminated. As the same



(a) Cost

(b) Utilization

(c) Average completion time of ChemApp requests

(d) Average waiting time of ChemApp requests

Fig. 3.12 Comparison of the performance metrics with A3R and without A3R

result is shown in Fig. 3.12d. The average waiting time of the application is dramatically reduced by applying the A3R policies.

3.8 Conclusions

As scientific applications become more complex, realizing workflows management for Cloud computing requires a number of challenges to be overcome. They include workflows application modeling, workflows scheduling, resource discovery, information services, data management and others. However, from the QoS perspective, two important barriers that need to be overcome are: (i) the integration of the workflows management system with resource management system and (ii) their scheduling on heterogeneous and distributed resources to meet user QoS demands.

In order to overcome the challenges in managing and scheduling workflows applications in Cloud, this chapter provides an efficient and flexible workflows scheduling mechanism which includes (i) functionalities for integrating workflows management with resource management system for QoS guarantee and (ii) various scheduling schemes for supporting different workflows applications, which have different QoS demands. We propose a cloud workflow scheduling mechanism incorporating with adaptive resource provisioning and the fine grained VM sprawl management heuristics for minimizing resource leasing cost. Uncertainty of future demands and prices of resources are taken into account to optimally adjust the tradeoff between on-demand and reserved instance costs. Especially, an adaptive resource reservation scheme (ARRS) reduces the total cost for leasing the VMs based on temporal marginal cost variation. Comparing to the optimal reservation capacity algorithm, it provides more exile and adaptive resource provisioning mechanism that easily adjusts the reserved VM within the payment duration of the reserved VM. On the other hand, we exploit three kind of effective resource sharing heuristics, which also decrease the total cost by increasing the utilization of the reserved VMs as well as already leased on-demand VMs.

The experimental results show, firstly, the proposed ARRS provides adaptivity of the resource reservation without the exact demand estimation. The comparison result from the optimal VM provisioning algorithm, the proposed ARRS is stable in the close to the optimal poinsts when time goes. On the other hand, the experimental results of A3R is performed on the real bio-chemical application. The results give us the feasibility of the A3R heuristics. By applying the A3R in the ChemApp application, we can reduce the total cost to maximum 80%. Moreover, the application completion is reduced upto 600 s.

References

1. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, Mapping abstract complex workflows onto grid environments, (2003)
2. D. Hollingsworth, WFMC: Workow reference model, Online PDF, Workow Management Coalition, Speci_cation, 1995, tC00-1003. [Online]. Available: http: //www.wfmc.org/standards/docs/tc003v11.pdf
3. D. Fernandez-Baca, Allocating modules to processors in a distributed system. IEEE Trans. Softw. Eng. **15**, 1427–1436 (1989)
4. M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of scienti_c workows in the askalon grid environment. SIGMOD Rec. **34**, 56–62 (2005)
5. H. Topcuouglu, S. Hariri, M.-Y. Wu, Performance-e_ective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**, 260–274 (2002)
6. M.R. Garey, D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness* (W. H. Freeman & Co., New York, NY, USA, 1979)
7. R. Sakellariou, H. Zhao, E. Tsiakkouri, M.D. Dikaiakos, Scheduling workows with budget constraints, in *Integrated Research in Grid Computing*, ed. by S. Gorlatch, M. Danelutto (Springer-Verlag, CoreGrid series, 2007)
8. W. Tan, Y. Fan, Dynamic workow model fragmentation for distributed execution. Comput. Ind. **58**(5), 381–391, (2007). [Online]. Available: http://dx.doi.org/10.1016/j.compind.2006.07.004
9. M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workows, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC ‘11. (ACM, New York, NY, USA), pp. 49:1–49:12. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063449

10. A. Benoit, Y. Robert, Complexity results for throughput and latency optimization of replicated and data-parallel workows, Algorithmica, **57**, 689–724 (2010). [Online]. Available: http://dx.doi.org/10.1007/s00453-008-9229-4
11. A. Sulistio, R. Buyya, A time optimization algorithm for scheduling bag-of-task applications in auction-based proportional share systems, in SBAC-PAD, pp. 235–242 (2005)
12. J. Yu, R. Buyya, A taxonomy of scientific workow systems for grid computing. SIGMOD Rec. **34**, 44–49 (2005)
13. F. Howell, R. Mcnab, Simjava: a discrete event simulation library for java, 51–56 (1998)
14. Equis, zaru inc, http://www.equispharm.com/html/html/main.html. [Online]. Available: http://www.equispharm.com/html/html/main.html
15. Gogrid, http://www.gogrid.com/. [Online]. Available: http://www.gogrid.com//
16. S. Chaisiri, B.-S. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, (2009), pp. 103–110

Chapter 4
A Cloud Broker System for Connected Car Services with an Integrated Simulation Framework

4.1 Introduction

At present, the mobile market accounts for the largest portion in IT industry, and its proportion is increasing rapidly. With the rapid increase, mobile services are also becoming bigger and more complex. Therefore, with the development of network technology such as 5G, there exist on-going research on mobile services that follows client-server models capable of overcoming the limitations of computational performance and storage in mobile devices.

Currently, the A connected car is the most emerging technology now in the mobile industry. The connected car refers to a vehicle provided with network connectivity. By 2025, SBD forecasts that 68% of new car sells will be correspond to [1]. At present, many automotive and IT vendors are developing the related services actively. Said services are classified into vehicle-to-vehicle (V2V) and vehicle-to-cloud (V2C). The V2V services are mainly based on the communication among vehicles while the V2C services consider the transactions between vehicles and a cloud. Both approaches enable each vehicle to overcome the hardware limitation and provide services that are difficult to handle in a single vehicle's onboard computer.

In this chapter, we focus on the V2C connected car services and means to handle a cloud broker system for their computation offloading. We start our discussion by classifying the V2C services into three categories: LDM, infotainment, and driving assistance. The detail description of each service type is handled with various examples and an analysis of their offloadability. We then describe the execution environment of the V2C services and an architecture of a cloud broker system to support the service execution. In the architecture, we mainly focus on the components related to the computational offloading, and several simple offloading approaches are introduced. To provide reliable evaluation of the service offloading, a simulation framework is presented for an integrated road traffic-network-cloud simulation. The framework supports end-to-end service simulation between

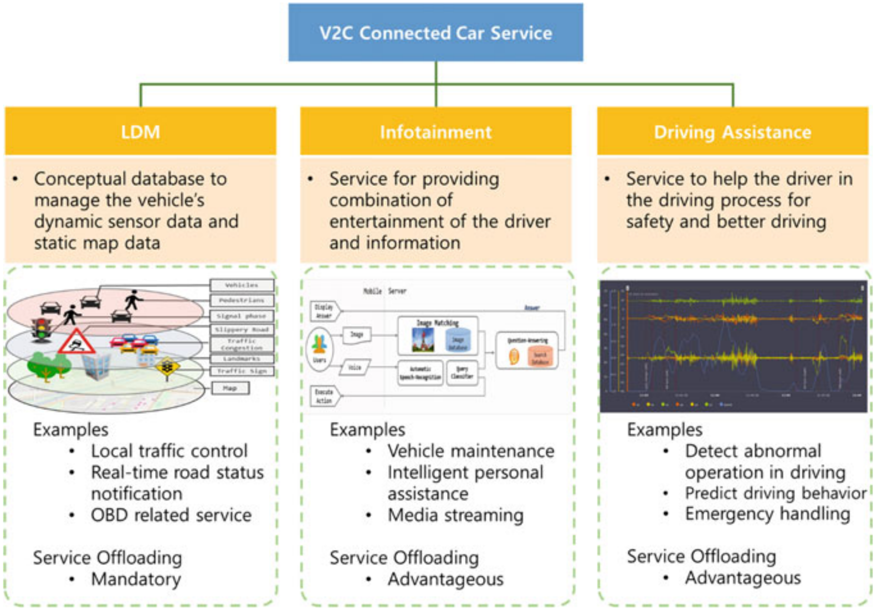
C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_4

vehicles and a cloud. To achieve it, each road traffic, network, and cloud simulations closely collaborate with each other. In addition, to address the difficulty of intuitively setting part of the simulation parameters, we also discuss a way to estimate the unknown parameters using the inverse simulation technique. It gives more reliability by increasing the accuracy of the integrated simulation. Finally, we present a proof-of-concept study to identify whether the cloud broker system enables proper execution the V2C services in terms of performance and cost.

4.2 A Cloud Broker System for V2C Connected Car Service Offloading

4.2.1 V2C Connected Car Service

Recent breakthroughs in communication technology have led to the emergence of connected cars. Providing network connectivity to vehicles, the concept of connected cars gives an opportunity to handle advantageously the vehicles as IT devices. As a result, connected cars are gaining great interest in the automotive industry, and many vendors are now developing innovative connected car services. We categorize of V2C connected car services as LDM [2–6], infotainment, and driving assistance, as shown in Fig. 4.1.


V2C Connected Car Service

LDM
- Conceptual database to manage the vehicle's dynamic sensor data and static map data

Examples
- Local traffic control
- Real-time road status notification
- OBD related service

Service Offloading
- Mandatory

Infotainment
- Service for providing combination of entertainment of the driver and information

Examples
- Vehicle maintenance
- Intelligent personal assistance
- Media streaming

Service Offloading
- Advantageous

Driving Assistance
- Service to help the driver in the driving process for safety and better driving

Examples
- Detect abnormal operation in driving
- Predict driving behavior
- Emergency handling

Service Offloading
- Advantageous

Fig. 4.1 The classification of V2C connected car services: LDM, infortainment, and driving assistance

Firstly, the LDM refers to the conceptual database based on interactive information management between vehicle sensor data and map data for road safety and traffic efficiency-related services [2]. The information consists of permanent static data, transient static data, transient dynamic data, and highly dynamic data as shown in Fig. 4.2. Since the sensor data in multiple vehicles should be aggregated and combined with map information in the cloud, the service offloading to a cloud is mandatory. An example of the LDMs is CARASSO [3]. As shown in Fig. 4.3, the CARASSO collects sensor data from BMW 7 series vehicles, and provides drivers dynamic map information using Amazon EC2. Drivenet Maps [5] also provide real-time dynamic maps for self-driving from 3D Lidar point clouds.

Secondly, the infotainment is the service that provide combination of entertainment and information for the driver. Vehicle maintenance, vision and voice based intelligent personal assistance and media streaming are examples of the infotainment services. In infotainment services, it is not necessary to communicate with the cloud, but it is possible to provide extensive functionality and processed information by using the information of the cloud as well as the vehicle's local information together. For example, you can only watch the movies that are stored in the local storage if the media service is not connected with the cloud, but you can watch any movies you want through media streaming service that connects with the cloud. At present, many automotive and IT vendors provide various systems such as Blue Link [7], iDrive [8], Uconnect [9], Android Auto [10], CarPlay [11], and so on. In addition, fundamental studies of an intelligent assistant as next-generation information systems have been proceeding vigorously [12, 13]. In case of Sirius [13], it provides an open end-to-end query and response applications for voice and vision-based intelligent personal assistance (IPA) (Fig. 4.4).

Finally, the driving assistance helps the driver to achieve safety and better driving. Services such as detecting abnormal operation in driving, predicting driving behavior, and emergency handling can be implemented as part of the driving assistance. Although driving assistance service such as hazard warnings can

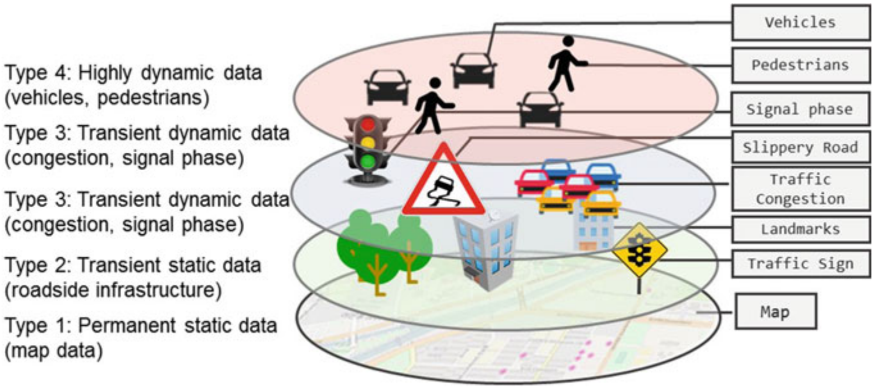

Type 4: Highly dynamic data
(vehicles, pedestrians)
Type 3: Transient dynamic data
(congestion, signal phase)
Type 3: Transient dynamic data
(congestion, signal phase)
Type 2: Transient static data
(roadside infrastructure)
Type 1: Permanent static data
(map data)
Vehicles
Pedestrians
Signal phase
Slippery Road
Traffic Congestion
Landmarks
Traffic Sign
Map

Fig. 4.2 A layered structure of LDMs [2]

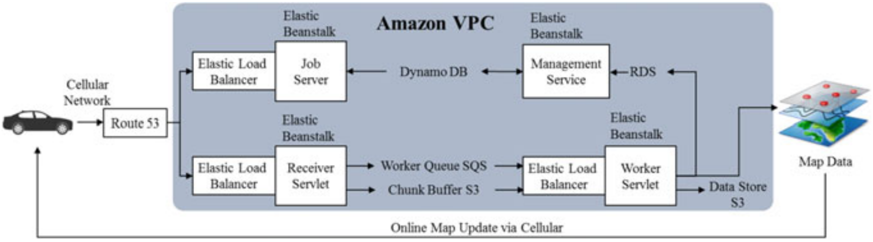


Fig. 4.3 Functional operations of CARASSO [3]

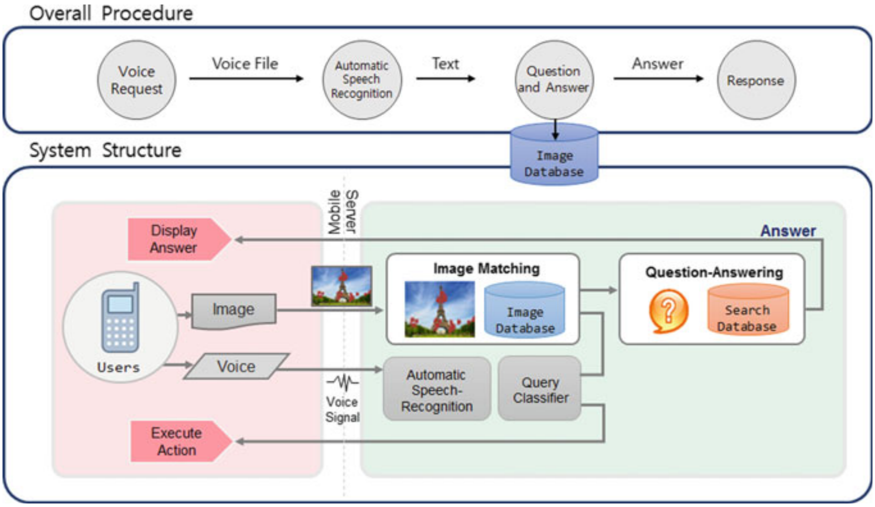


Fig. 4.4 Functional operations of Sirius [13]

be implemented only using vehicle's local information such as sensor data or front view camera image, it can be more accurate and provide more various extensive services by using the information of the cloud such as aggregated big data. There have been a number of works for driving assistance systems such as [14–17]. The purposes of the driving assistance systems is divided into (1) detecting abnormal operation in vehicle driving and (2) predicting driver's behaviors based on sensing data. As an example of the first issue, Kumar et al. [14] and Ashok et al. [15] presented a framework for computation offloading of vehicular applications. In [14], the authors proposed a cloud-assisted system for autonomous driving to provide computation offloading of sensor data analytics. The system enables vehicles to avoid obstacles, pedestrians and other vehicles as well as properly handle emergencies. In [15], the authors proposed a novel architecture to adaptively manage computation offloading and evaluated its prototype using computer vision applications like motion hand gesture recognition and traffic light and sign recognition. For the second issue, Pivotal [16] and Blind Motion [17] provides deep

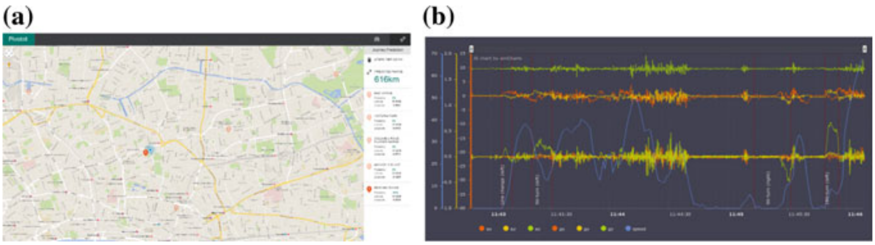


Fig. 4.5 Dashboards for the deep learning systems to **a** predict destinations of drivers [16] and **b** detect maneuvers of vehicles [17]

learning systems to respectively predict destinations of drivers and detect maneuvers of vehicles (Fig. 4.5).

4.2.2 An Architecture of the Cloud Broker System with Service Offloading Strategies

As shown in Fig. 4.6, we consider the V2C service environment consisting of vehicles, base stations, a cloud broker and a cloud. In the figure, the vehicles are supposed as end-users of V2C services. Even though the vehicles are equipped with internal processors, it can be more efficient to offload the services to more computationally rich resource by the following reasons. First, it allows overcoming

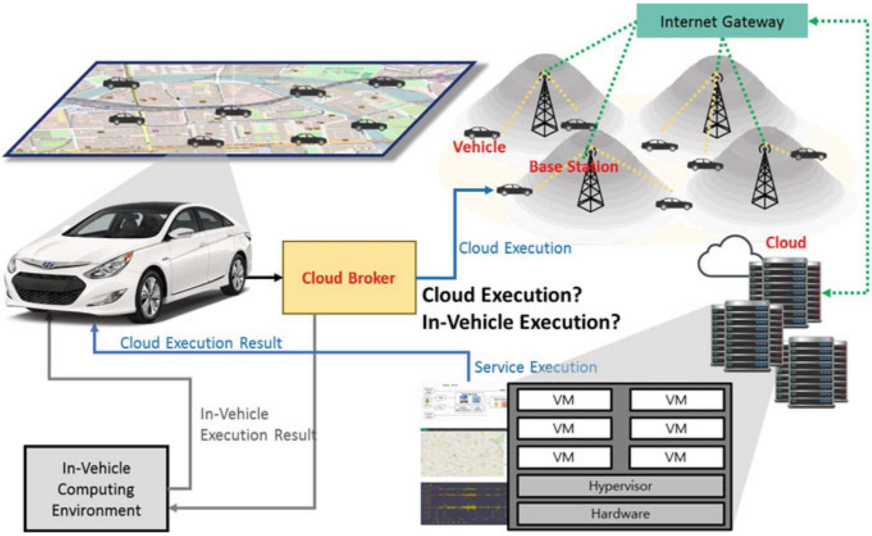


Fig. 4.6 An example of a V2C service environment

computing and storage limitations of vehicles. Second, it can achieve cost saving by relaxing the needs to maximize the resource capacity in vehicles. Third, it allows reducing the battery consumption of in-vehicle computations. Fourth, it enables cloud-centric services that are not possible in a single vehicle. Thus, we consider the computation offloading to the cloud in the service environment. The cloud broker determines whether to offload the vehicle requested services or not. If the remote execution is decided, the services are executed in the cloud instead of the vehicles.

Figure 4.7 shows the architecture of cloud broker systems to effectively offload V2C connected car services. Even though more functionality should be included to support the whole service operations, the figure illustrates functions related to the computation offloading. The cloud broker, as the linkage between the vehicles and the cloud, manages the whole execution of the services. An advantageous offloading strategy can vary depending on each service. Therefore, the cloud broker systems consider five offloading strategies by the characteristics of the services as follows.

- No offloading

No offloading refers to a method of conducting service executions only in vehicles without using the cloud. Here, the service completion time is equal to in-vehicle computation time. Non-offloadable services should adopt this strategy mandatorily. In addition, the strategy is advantageous for services where in-vehicle execution is more efficient than the service offloading.

- All offloading [18–23]

All Offloading refers to a method of offload every available service execution to the cloud. Here, the service completion time is the same as the sum of cloud computation time and data transmission/reception time. This strategy is mandatory for the cloud-centric services. Also, it is advantageous to adopt the strategy for complex services in which in-vehicle execution is highly burdensome.

- Network-adaptive offloading [24–28]

Network-adaptive offloading refers to a method for determining whether to offload services adaptively to real-time network performance. If $\widehat{T}_{tx}, + \widehat{T}^{cloud}_{comp} + \widehat{T}_{rx} < \widehat{T}^{veh}_{comp}$ where $\widehat{T}_{tx}$, $\widehat{T}^{cloud}_{comp}$, $\widehat{T}_{rx}$, and $\widehat{T}^{veh}_{comp}$ are respectively the estimated data transmission time, cloud computation time, data reception time, and in-vehicle computation time, the service offloading is executed because the service completion time with the computation offloading is shorter than in-vehicle computation time. Otherwise, in case of $\widehat{T}_{tx}, + \widehat{T}^{cloud}_{comp} + \widehat{T}_{rx} \geq \widehat{T}^{veh}_{comp}$, the service offloading does not occur because the service completion time with the computation offloading is longer than in-vehicle computation time.

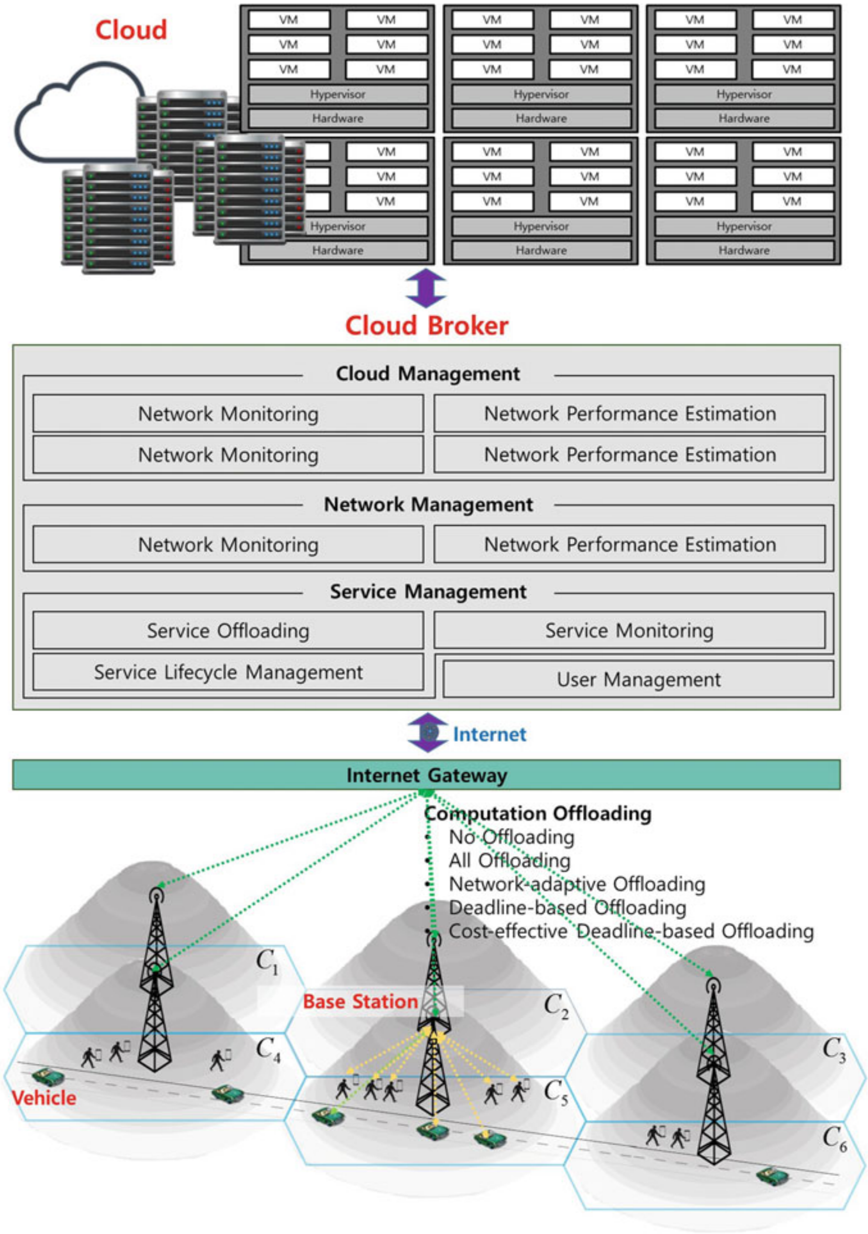


Fig. 4.7 An architecture of the cloud broker system for V2C connected car service offloading

- Deadline-based offloading [29–32]

Deadline-based offloading refers to a method to determine the service offloading depending on deadline satisfaction. At first, every service is offloaded to the cloud. If the service execution in the cloud satisfies the deadline, the service is provided with the cloud continuously. Otherwise, the next service execution is conducted in vehicles. Likewise, if the service execution in the vehicles does not meet the deadline, the next service trial is processed in the cloud.

- Cost-effective deadline-based offloading [33–39]

Cost-effective deadline-based offloading is an extension of deadline-based offloading. In this method, whether to execute is determined to minimize cost while guaranteeing to meet the deadline as described in the following optimization problem:

$$\min C_{comp}^{cloud} + C_{net}^{cloud} \tag{4.1}$$

$$\begin{aligned} s.t.\ & \hat{T}_{tx} + \hat{T}_{comp}^{cloud} + \hat{T}_{rx} \leq deadline\ or \\ & \hat{T}_{comp}^{veh} \leq deadline. \end{aligned} \tag{4.2}$$

4.3 An Integrated Road Traffic-Network-Cloud Simulation Framework for V2C Connected Car Services Using a Cloud Broker System

4.3.1 An Overview

Figure 4.8 shows an overview of the integrated road traffic-network-cloud simulation framework. In the framework, given V2C applications, the network and the cloud simulations operate tightly coupled based on a vehicular trace obtained by road traffic simulation. When the simulation begins, the V2C environment is created based on the simulation setup, and vehicles start to move along the trace. For each vehicle, the task execution for every application is requested at the corresponding request interval. Whether to offload the execution is determined by the offloading policy of said application. If it is determined not to offload, the applications are executed in the vehicle itself. Otherwise, the applications are offloaded to clouds for their execution. After the execution, the results are returned to each vehicle. The simulation framework mimics the entire end-to-end transactions of the applications and estimates the cost and performance during each step. We note that the integrated simulation framework is description based on a part of [40].

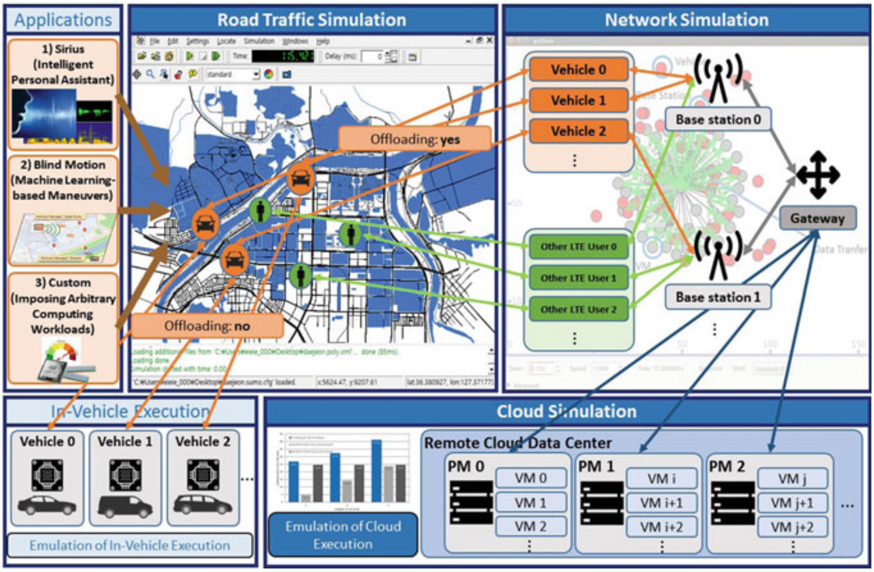


Fig. 4.8 An overview of the integrated road traffic-network-cloud simulation

4.3.2 An Architecture of the Integrated Simulation Framework

Figure 4.9 shows an overall architecture of the simulation framework. In the framework, the simulation gateway is responsible for managing the overall operations of the simulation. The overall operations are composed of the following five steps as shown in Table 4.1.

- Simulation setup: In this step, the simulation users enable to select the region that would be simulated and manipulate the simulation parameters.
- Task execution emulation: Given the user-specified cloud resource configuration, the simulation gateway carries out cloud resource and task assignment in a cloud datacenter. Based on the assignment result, the emulation of task execution is conducted to mimic the computational behaviors of the applications to be evaluated.
- Main simulation parameter setting: Based on the first and the second step, the simulation gateway determines the final parameters for the main simulation.
- Main simulation: Requested by the simulation gateway, the main simulation is conducted by the simulation controller to evaluate the end-to-end performance and cost of the applications.
- Result analysis: After the main simulation, the simulation gateway provides the simulation results to the users in a graph form. The simulation results include service completion time, service costs, and various network performance.

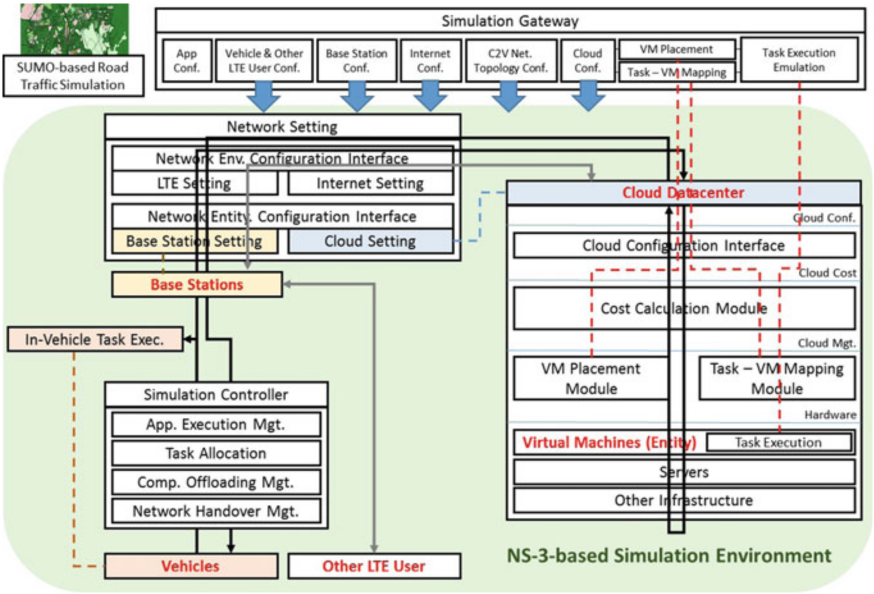


Fig. 4.9 An overall architecture of the simulation framework

Table 4.1 Simulation steps

Request step	URI format	Step description
Simulation setup	{Simulation gateway address}:{port}/sim_setup/{JSON text for the simulation setup}	Select the region to be simulated and manipulate the simulation parameters
Task execution emulation	{Simulation gateway address}:{port}/sim_emulation	Emulate the task execution according to the resulted cloud resource and task assignment in a cloud datacenter
Main Simulation parameter setting	{Simulation gateway address}:{port}/sim_preprocessing	Determine the final parameters for the main simulation based on the simulation configuration and task execution emulation
Main simulation	{Simulation gateway address}:{port}/sim_start	Run the main simulation
Result analysis	{Simulation gateway address}:{port}/sim_results	Provide the graphical summarization of the simulation results

- Simulation setup

As the first step, the simulation gateway enables the simulation users to specify the V2C environment and its region. The users can select the region using OpenStreetMap [41] with its longitude and latitude. After the selection, a vehicle mobility scenario is created using SUMO [42]. In addition, the simulation framework provides the users with various parameters to manipulate as shown in

Table 4.2. The parameters are related to the entire simulation, vehicles, other LTE users, base stations, Internet, a cloud datacenter, applications, and cloud resource configurations of each application.

- Cloud resource configuration

The simulation framework allows the simulation users to configure VM placement and task-VM mapping in a cloud datacenter. The VM placement and the task-VM mapping determine how to locate VMs to each PM and how to assign tasks to each VM respectively. In the simulation framework, the cloud datacenter is supposed to consist of clusters with homogeneous PMs for each application only. In addition, we consider VMs of the same flavor in the simulation.

– VM placement

At present, we implemented three placement strategies as shown in Table 4.3. VMP-CON is a strategy that maximally consolidates VMs. VMP-MAN is a strategy in which the simulation users can configure the number of the co-located VMs in a PM. VMP-FAIR, as opposed to VMP-CON, is a strategy that locates a fair number of VMs in a PM. Under the VMP-CON, we can achieve the reduction of power consumption by minimizing the number of active PMs. However, this strategy can increase the possibility of performance degradation caused by performance interference. The VMP-FAIR, on the other hand, can minimize performance interference by maximally dispersing VMs, but the corresponding power consumption increases.

– Task-VM mapping

At present, VMM-FAIR has been implemented in the simulation framework. The strategy allocates equal number of tasks to VMs for each application.

- Mimicking the task execution

In the simulation framework, the task execution is mimicked based on careful emulation using real cloud testbed. For the emulation, a single compute node in the testbed is utilized, and it operates by actually executing the applications to be evaluated. Figure 4.10 shows the emulation procedure of each application. Given the results of the VM placement, the emulation module creates the determined number of VM instances in the compute node. Then, the number of tasks determined from the task-VM mapping is executed in each VM instance via the emulation agent. After the emulation is finished, the execution results are returned to the emulation module. The task execution time is given by $N\left(e_{VM} \cdot \mu, e_{VM}^{2} \cdot \sigma^{2}\right)$ and $N\left(e_{VEH} \cdot \mu, e_{VEH}^{2} \cdot \sigma^{2}\right)$, for clouds and vehicles respectively. μ and σ denote the mean and the standard deviation of the aggregated results.

- Main simulation

The main simulation works through an online integration link of road traffic, network, and cloud simulation. As a preliminary stage of the main simulation, the V2C

Table 4.2 Simulation parameters

Parameter	Description
simDuration	Simulation duration (s)
v2cModel.vehicle.number	Number of vehicles
v2cModel.vehicle.txPower	Transmission power of vehicles (dBm)
v2cModel.vehicle.noiseFigure	Noise figure of vehicles (dB)
v2cModel.vehicle.maxMobility	Maximum speed of vehicles (m/s)
v2cModel.vehicle.executionPerformanceConstant (e_{VEH})	Relative performance constant of vehicles
v2cModel.otherLTEuser.number	Number of other LTE users
v2cModel.otherLTEuser.mobility	Speed of other LTE users (m/s)
v2cModel.otherLTEuser.inptDataSize	Input data size of other LTE users (byte)
v2cModel.otherLTEuser.outputDataSize	Output data size of other LTE users (byte)
v2cModel.otherLTEuser.requestInterval	Request interval of other LTE users (s)
v2cModel.baseStation.number	Number of base stations
v2cModel.baseStation.scheduler	LTE Mac scheduler
v2cModel.baseStation.pathLossModel	Path loss model
v2cModel.baseStation.UlBandwidth	Uplink bandwidth of base stations (MHz)
v2cModel.baseStation.DlBandwidth	Downlink bandwidth of base stations (MHz)
v2cModel.baseStation.UlEarfcn	Uplink Earfcn of base stations (MHz)
v2cModel.baseStation.DlEarfcn	Downlink Earfcn of base stations (MHz)
v2cModel.baseStation.txPower	Transmission power of base stations (dBm)
v2cModel.baseStation.noiseFigure	Noise figure of base stations (dB)
v2cModel.baseStation.handoverAlgorithm	Handover algorithm
v2cModel.internet.dataRate	Data rate of Internet (Gb/s)
v2cModel.internet.MTU	MTU of Internet
v2cModel.internet.delay	Delay of Internet (s)
cloudModel.datacenter.serverCount	Number of compute nodes
cloudModel.datacenter.wattsPerServer	Provisioned power in compute nodes (W)
cloudModel.datacenter.PUE	PUE
cloudModel.datacenter.powerCost	Hourly electiricity price ($)
cloudModel.costPolicy	Cost policy for VM instances

(continued)

Table 4.2 (continued)

Parameter	Description
cloudModel.vm.executionPerformanceConstant (e_{VM})	Relative performance constant of VMs
app.{app}.useOrNot	Whether or not to use {app} in the simulation
app.{app}.offloadingStrategy.strategy	Service offloading strategy of {app}
app.{app}.inputDataSize	Input data size of {app} (byte)
app.{app}.outputDataSize	Output data size of {app} (byte)
app.custom.workloadSize	Workload size of *Custom* if it is determined to use
app.{app}.requestInterval	Request interval of {app} (s)
app.{app}.deadline	Deadline of {app} (s)
app.{app}.vmNumber	Number VMs in the cluster for {app}
app.{app}.vmFlavor	Flavor of VM instances for {app}
app.{app}.vmpStrategy.strategy	Cloud resource placement strategy for {app}
app.{app}.vmpStrategy.maxVMinPM	Maximum number of co-located VMs in a compute node for {app}
app.{app}.vmpStrategy.manVMinPM	Number of co-located VMs in a compute node for {app} if the cloud resource placement strategy is *Man*
app.{app}.vmmStrategy	Task-VM mapping strategy for {app}

Table 4.3 VM placement strategies for VM placement module

Scheme	Description	Pros
VMP-CON	Maximize VM Consolidation	Power efficient
VMP-MAN	Configure the number of VMs located in each PM according to the simulation setup	Balance the advantages of the VMP-CON and the VMP-FAIR
VMP-FAIR	Maximize VM dispersion	Less performance interference

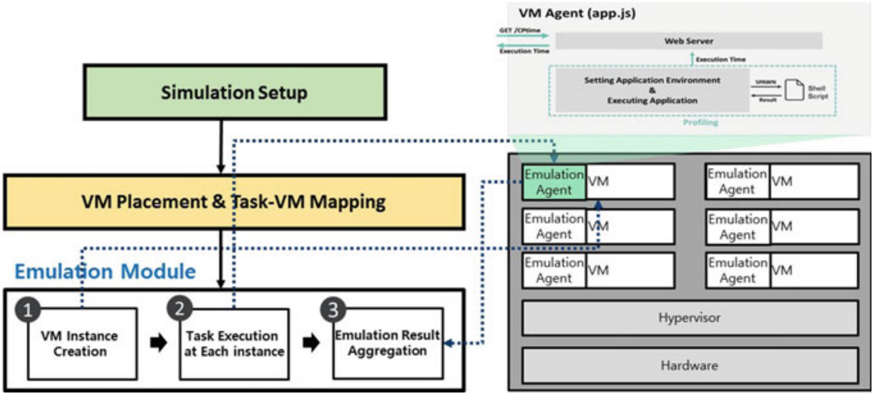


Fig. 4.10 The emulation procedure of each application

environment is firstly configured by creating the V2C components (UEs—vehicles and other LTE users, base stations, EPC, and VMs) and establishing the connection among them. The vehicular trace is also generated from the vehicle mobility scenario using SUMO [42]. The main simulation is managed by the simulation controller according to each simulation instance as shown in Fig. 4.11. The simulation instance refers to the service path of each application for each vehicle and other LTE users. Based on the simulation instance, an end-to-end simulation is carried out to handle the entire process of applications including the service request, execution, and response. The whole network mechanisms of the simulation framework is based on ns-3 [43] and Lena [44]. At present, network transactions such as data transfer in each simulation instance, have been implemented to consider the LTE model. In addition, as stated above, the task execution in each simulation instance is mimicked using an emulation based on the user-specified cloud resource configuration. Note that we assume that only data transfer occurs without the task execution in the case of other LTE users.

- Result analysis

In the simulation framework, simulation results are summarized as service completion time, service costs, and various network performance indicators. Figure 4.12 shows an example of the selected simulation results.

Figure 4.12a, b, c consider service completion time (s). The service completion time includes data transmission time, data receiving time, and task computation time. The task computation can be either in clouds or in vehicles. Figure 4.12a, b, c respectively show the average application processing time with respect to vehicle IDs, application processing time of a selected vehicle with respect to application processing trials, and Gantt chart which describes application processing status of each vehicle with respect to time.

Figure 4.12d shows an analysis of cloud provider's monthly profit, and it is calculated as the monthly total cloud service cost of all vehicles—the monthly total SLA penalty cost—the monthly PM power cost. The figure describes the simulation results assuming that vehicles are provided with services 6 h a day. The cloud service cost refers to cloud consumers' (service providers) service charge to cloud providers. The cloud service cost includes both computing and network resource usage cost. The simulation framework utilizes cost policies of public cloud providers such as Amazon EC2 [45]. The SLA penalty cost refers to the penalty charge if the cloud provider violates the SLA. In the figure, the SLA penalty costs are shown

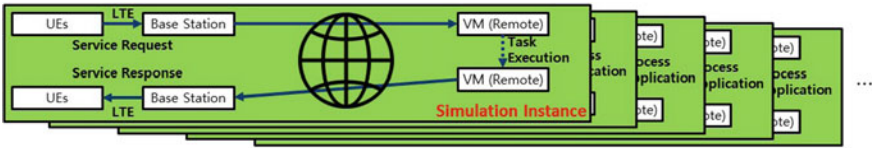


Fig. 4.11 Simulation instances

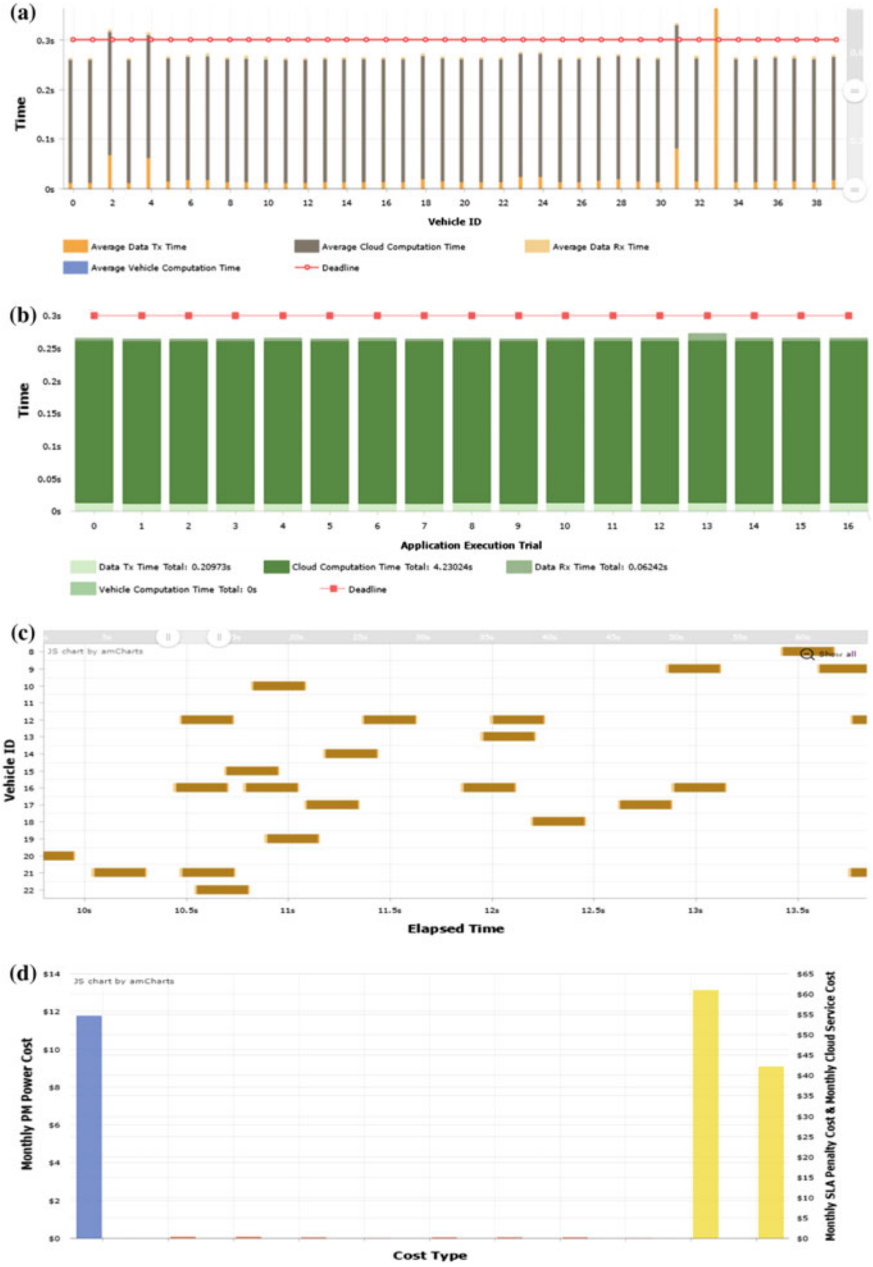


Fig. 4.12 An example of the selected simulation results. **a** Average service completion time (for all vehicles). **b** Service completion time (for each vehicle). **c** Service completion time (Gantt chart). **d** Monthly profit analysis of the cloud provider. **e** Handover count. **f** Average RLC downlink throughput. **g** Average RLC uplink throughput. **h** Average downlink SINR. **i** Average uplink SINR. **j** PDRC downlink throughput. **k** PDRC uplink throughput. **l** Transmitted packet number. **m** Received packet number. **n** Transmitted byte size. **o** received byte size. **p** Packet loss ratio

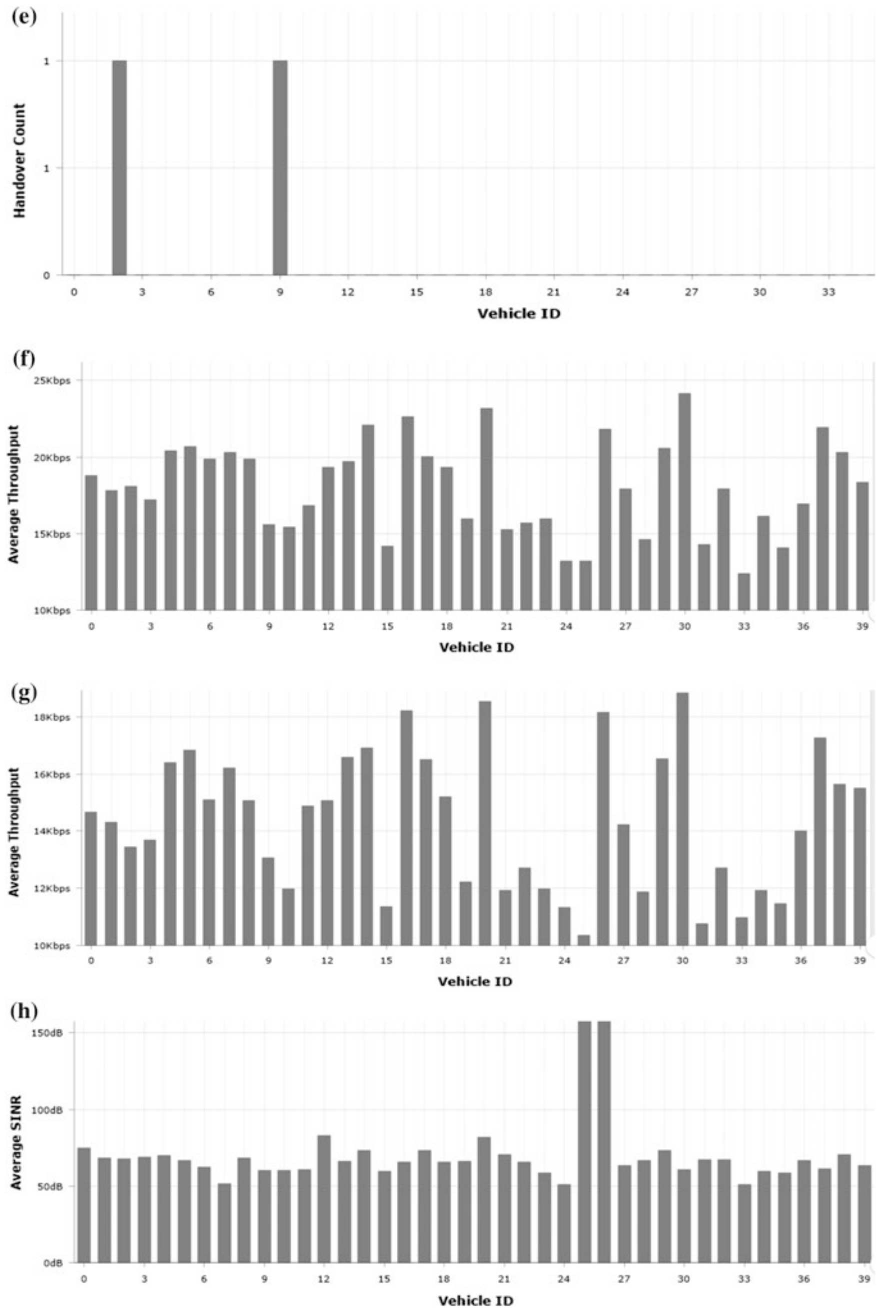


Fig. 4.12 (continued)

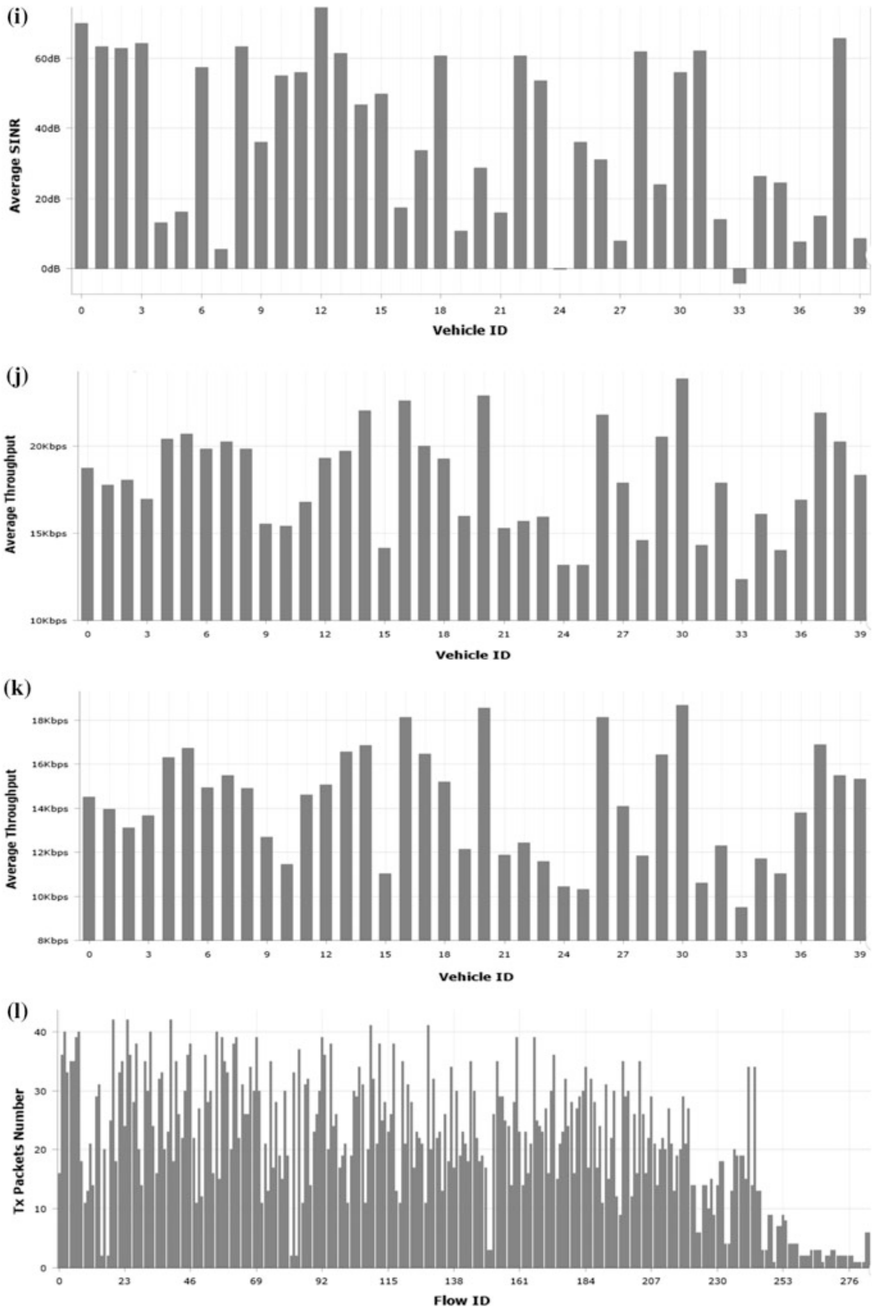


Fig. 4.12 (continued)

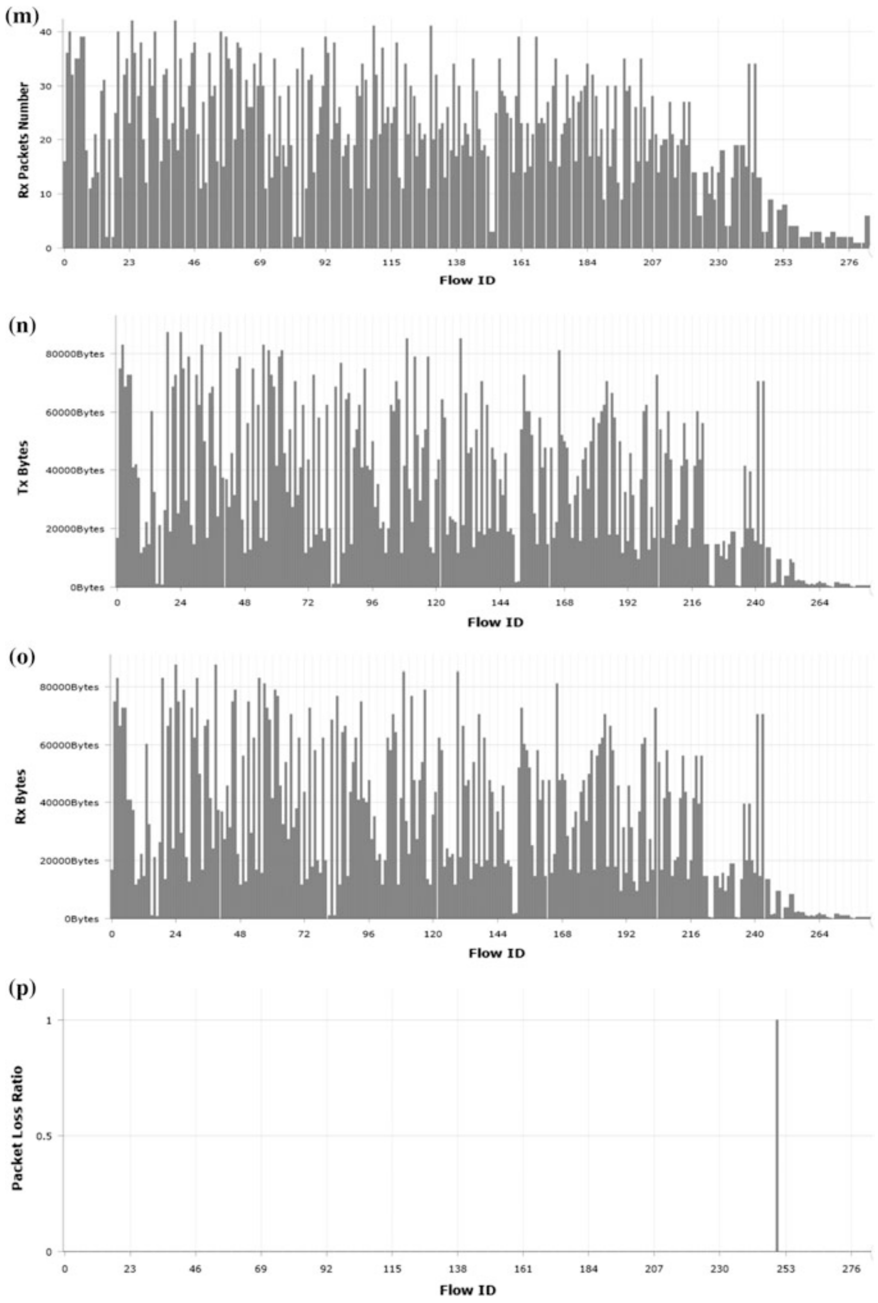


Fig. 4.12 (continued)

under the following cases: (degradation-proportional, on-demand), (degradation-proportional, reserved), (Amazon EC2 [45], on-demand), (Amazon EC2, reserved), (Google Compute Engine [46], on-demand), (Google Compute Engine, reserved), (Microsoft Azure [47], on-demand), (Microsoft Azure, reserved). We note that the elements in the 2-tuple refers a SLA penalty cost strategy and a VM pricing type respectively. As for VM pricing types, we consider both on-demand and reserved types. On-demand and reserved types of VMs have relatively short billing time unit (BTU) (e.g., an hour) and long BTU (e.g., a month, an year) respectively. Therefore, service charges of reserved types of VMs are cheaper than that of on-demand types in the same period. The PM power cost refers to the operating cost of PMs in a cloud datacenter, and it is based on Hamilton's model [48].

Figure 4.12e, f, g, h, i, j, k handle network performance with respect to vehicle IDs. Figure 4.12e shows the handover count. The handover refers to connecting a UE to a closer and more signal-strong base station if the signal strength between them gets weaker. Figure 4.12f, g show the average downlink and uplink throughput (Kbps) of LTE radio link control (RLC) protocols respectively. Figure 4.12h, i show the average downlink and uplink SINR (dB) respectively. Figure 4.12j, k show the average downlink and uplink throughput (Kbps) of LTE packet data convergence protocol (PDCP). The PDCP refers to a protocol to provide header compression and decompression of IP data streams, transfer of user data, and maintenance of PDCP sequence numbers for radio bearers [49]. Figure 4.12l, m, n, o, p handle network performance with respect to flow IDs. The figures handle the number and size of transmitted and received packets, and packet loss ratio respectively.

4.3.3 *The Extension of the Integrated Simulation Based on the Inverse Simulation Technique*

Even though the integrated simulation framework provides an effective way to evaluate the V2C connected car services, the simulation users could encounter difficulties when specifying some simulation parameters intuitively due to lack of expertise or familiarity with the options. To address this issue, we present a way to improve the accuracy of the simulation integration (see Fig. 4.13). The first step is to identify the unknown parameters, which mainly include the environmental variables. Based on the available real-world measurement of the simulation outputs in the region, the values of the unknown variables are then estimated using the inverse simulation technique. Inverse simulation refers to an inverse process of direct simulation, and it enable to identify the simulation inputs that are difficult to be determined with the given simulation outputs. At present, the inverse simulation is widely used in various research areas, and there also have been a lot of recent works for automotive technology such as [50–54]. The measurement ensures

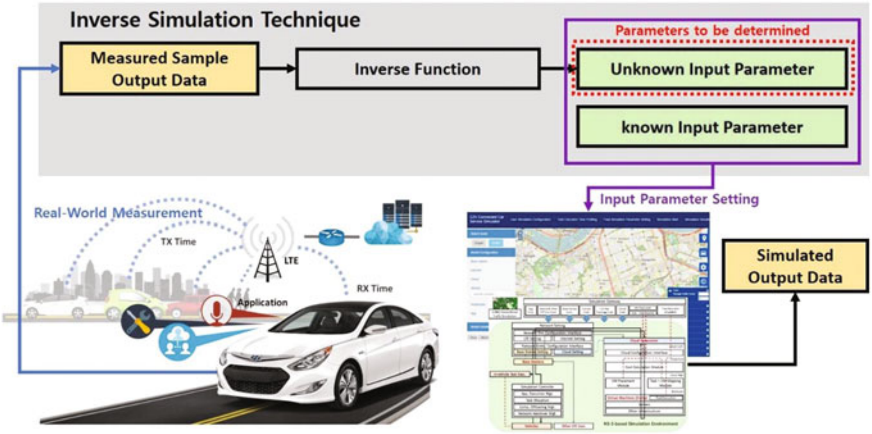


Fig. 4.13 A procedure for the identification of the unknown simulation parameters

accuracy in the estimation due to its results can be considered as the most accurate simulation results. Finally, the estimated values are utilized in the simulation setup with the remaining user-specified simulation parameters.

- The estimation of the unknown simulation parameters and its validation
  Assuming performance of the backbone network is almost constant, we consider data rate and delay of Internet as the unknown inputs among the simulation parameters described in Table 4.2. So far, we describe the detail procedure of estimate the unknown inputs from the measured service performance.
  For the measurement, we built a service prototype of the Blind Motion [17] as shown in Fig. 4.14. The application detects various driving maneuvers such as lane changes, obstacle avoidances, overtakes, direction changes, U-turns by utilizing the sensor data generated by smartphones in the vehicle.

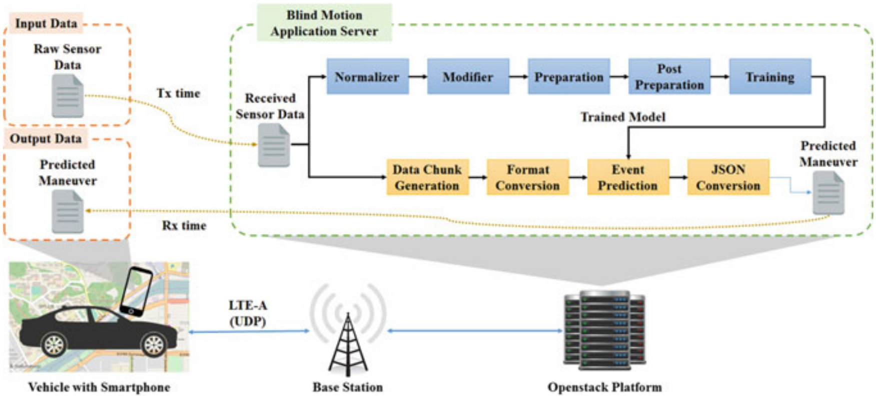


Fig. 4.14 A service prototype

For the detection, the application needs to learn the relationship between the sensor data that actcs as the input data and the driving maneuvers that correspond to the output data, using the deep neural network with three hidden layers in advance. Before training the model using sensor data, the raw sensor data is preprocessed. Firstly, the raw sensor data is normalized. After the normalization, in the modifier module, each the normalized data element is multiplied with the predefined weight to smoothen the data. In the preparation module, several data sets in csv file format are generated from the smooth data and event json file of various vehicles. The training and validation data set in both csv and mat file formats are generated in the post preparation module by aggregating several data sets generated in the preparation module. After the training data set is generated, the deep neural network model is trained using the training data set and then trained model is generated in mat file format.

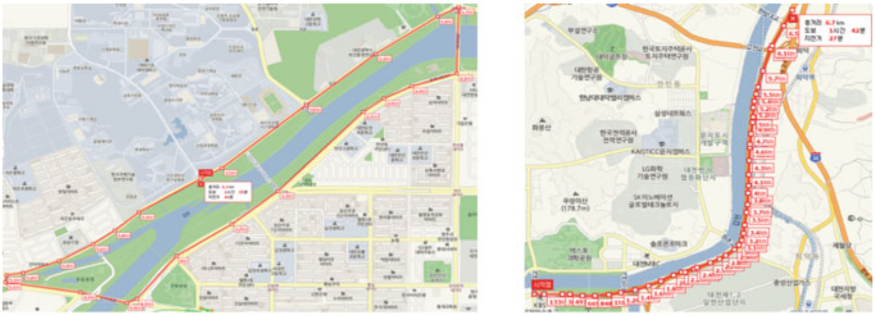
When the service is started, vehicles (i.e. the end-users), transmit the sensor data to the cloud at given interval. After receiving the data from each vehicle, the application server detects the driving maneuvers in the following steps. First of all, the application server generates chunks of sensor data from the received data. Secondly, the application server converts chunks of sensor data in csv file format to the file in mat file format. Third, the application server predicts events using the chunks of sensor data in mat file format and the trained model. Finally, the predicted events in csv file format are converted to the json format file. After the detecting the driving maneuvers in the application server is completed, the application server transmits the final predicted event data to each corresponding vehicle. In the prototype, input data is set to about 8 KB, and the corresponding output data size is set to 0.1 KB.

Table 4.4 shows the hardware specifications of the service prototype. In the prototype, a smartphone have been utilized as the client device, and the client part was implemented in it. For the application servers, we have utilized VMs, and they are built in the OpenStack [55] compute node that has eight cores of Intel® Xeon® CPU E5-2650 v2 @ 2.60 GHz model and 16 GB RAM.

After installing the prototype on the smartphone, by driving the vehicle equipped with it, we have measured the network performance to estimate the unknown parameters. For the region of interest (ROI), we selected two routes nearby Gapcheon stream as illustrated in Fig. 4.15. For each driving route, the average driving speed was set to 60 km/h (=16.67 m/s) and 80 km/h

Table 4.4 Hardware specifications of the service prototype

Role	Product	Specification
Client device	LG G4 Smart Phone	• Hardward: Qualcomm Snapdragon 808, 3 GB LPDDR3 RAM, 32 GB eMMC • Telecommunication: 3Band LTE-A • Operating System: Android 6.0
Cloud platform	Openstack [55] Compute Node	• Hardware: Intel Xeon E5620 2.40 GHz (2 Cores), 2 GB RAM, 100 GB HDD • Operating System: Ubuntu 14.04



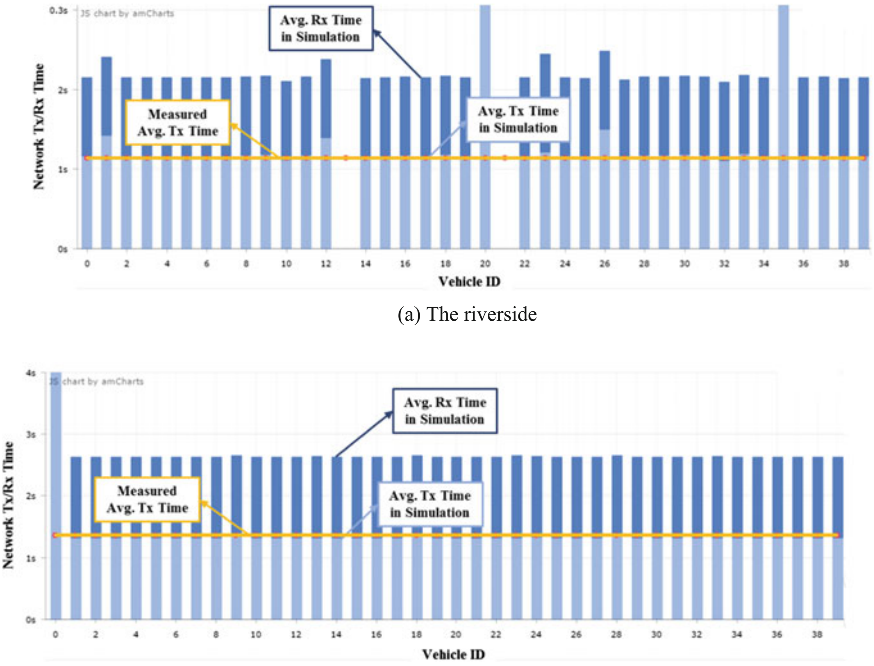
(a) The riverside

(b) The expressway

Fig. 4.15 Driving routes illustrated by OpenStreetMap [41] and NAVER Maps API [56]

(=22.22 m/s) respectively. While driving, the execution requests were generated and measured every five second.

Figure 4.16 shows the comparison between measured data transmission time and its simulated value obtained from the parameter estimation. The yellow line denotes the measured data transmission time, and its values are 114.145 and



(a) The riverside

(b) The expressway

Fig. 4.16 The comparison between measured data transmission time and its simulated value obtained from the parameter estimation using the PIS technique

Table 4.5 The selected simulation parameters and their values

Description		Value
Simulation duration		60
Vehicle	Number	[40, 60, 80]
	Maximum speed (m/s)	–
	e_{VEH}	1
Other LTE user	Number	100
	Speed (m/s)	1
	Input/Output data size (byte)	512/512
	Request interval (s)	Exp(0.1)
Base station	Number	20
	Handover algorithm	A2-A4-RSRQ
Internet	Data rate (Gb/s)	116
	MTU	30,000
	Delay (s)	0.095
Service (Blind Motion)	Offloading strategy	Deadline-based offloading
	Input/Output data size (byte)	8187/116
	Request interval (s)	Exp(0.2)
	Deadline (s)	4.981
	Number of VMs in the cluster	40
	Flavor of VM instances	t2.small
	Maximum number of co-located VMs in a compute node	10
	Number of co-located VMs in a compute node	4
Cloud datacenter	e_{VM}	1

1357.107 ms for the riverside and the expressway respectively. The light blue bars denote the simulated values, and they have been derived via multiple simulation trials with various settings of the unknown parameters. For the rest of the simulation parameters, the simulation setup was conducted using the values described in Tables 4.5 and 4.6 supposing that the number of vehicles is 40. As shown in the figure, the simulated values with the estimated unknown parameters are quite close to the measured values for both the riverside and the expressway. We can see that 82.5 and 97.5% of the vehicles show almost the same valueto that of the measurements.

Table 4.6 The estimated task execution time

		Number of vehicles		
		40	60	80
VMs	Avg.	2.832	3.006	3.154
	Std.	0.259	0.516	0.617
Vehicles	Avg.	3.0		
	Std.	0.5		

4.3.4 A Proof-of-Concept Study of the Service Execution with the Cloud Broker System

In this subsection, we present a proof-of-concept study of the service execution with the cloud broker system. To achieve this, we carried out an example of the simulation using the integrated simulation framework. The simulations have been conducted for the Blind Motion in three cases where the number of vehicles were 40, 60, and 80 respectively. We utilized the same area with the previous subsection for the simulation region. Figure 4.17 shows the simulation region illustrated by SUMO [42].

- Simulation setup

  Table 4.5 shows the selected simulation parameters and their values for the simulation. We note that the internet data rate and delay have been determined by the parameter estimation, and the parameters specifying each vehicle and base station were set based on the default values in ns-3 [43] and Lena [44]. Table 4.6 shows the estimated task execution time based on the results of the task execution emulation of the Blind Motion. For the emulation, we utilized the same compute node described in Table 4.4.

- Simulation result

  Figure 4.18 illustrates the average completion time. In the figure, we can see that the number of vehicles whose completion time is longer than the SLA deadline increases with respect to their total number. The results show that the average SLA satisfaction ratios throughout the simulation duration are 94.9, 61.8, and 48.6% for the cases of 40, 60, and 80 vehicles respectively. The main rationale is that when the number of vehicles increases, the more vehicles utilize the limited cloud resources, and thus the presence of performance interference among VMs. In this case, to increase the SLA satisfaction ratio, more cloud resources should be provided while the corresponding cost would be increased.

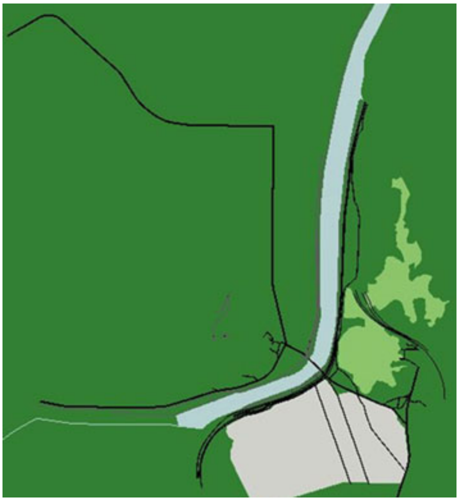


Fig. 4.17 The simulation region illustrated by SUMO [42]

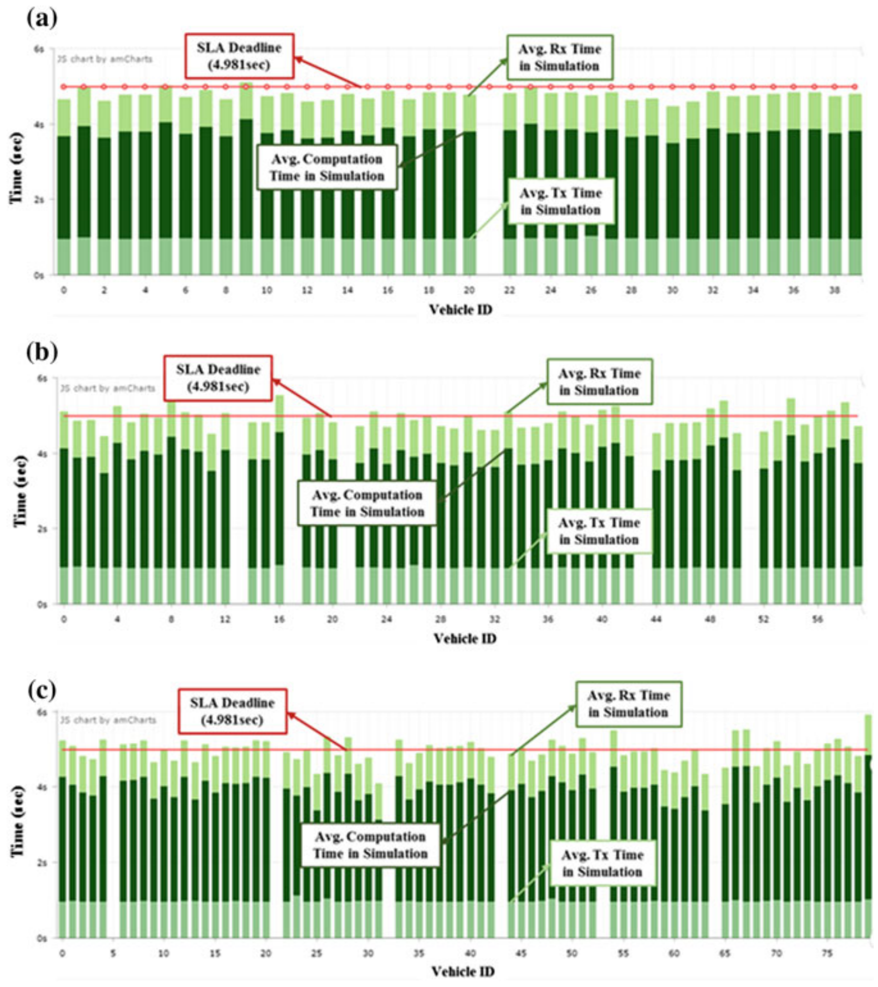


Fig. 4.18 The average completion time by manipulating the number of vehicles as **a** 40, **b** 60, and **c** 80

Figure 4.19 shows a monthly profit analysis of the cloud provider assuming the service is executed during 6 h every day. As shown in the figure, the monthly SLA penalty cost increases with the growth of the number of vehicles. It is because the average SLA satisfaction ratio is inversely proportional to the number of vehicles as shown in Fig. 4.18. However, the monthly cloud usage cost does not vary with the number of vehicles because the number of VMs and usage time is equivalent. Moreover, the monthly operating cost of compute node tends to increase with respect to the vehicle amount. The reason for this is that VM utilization increases as the number of vehicles increases, and causes greater

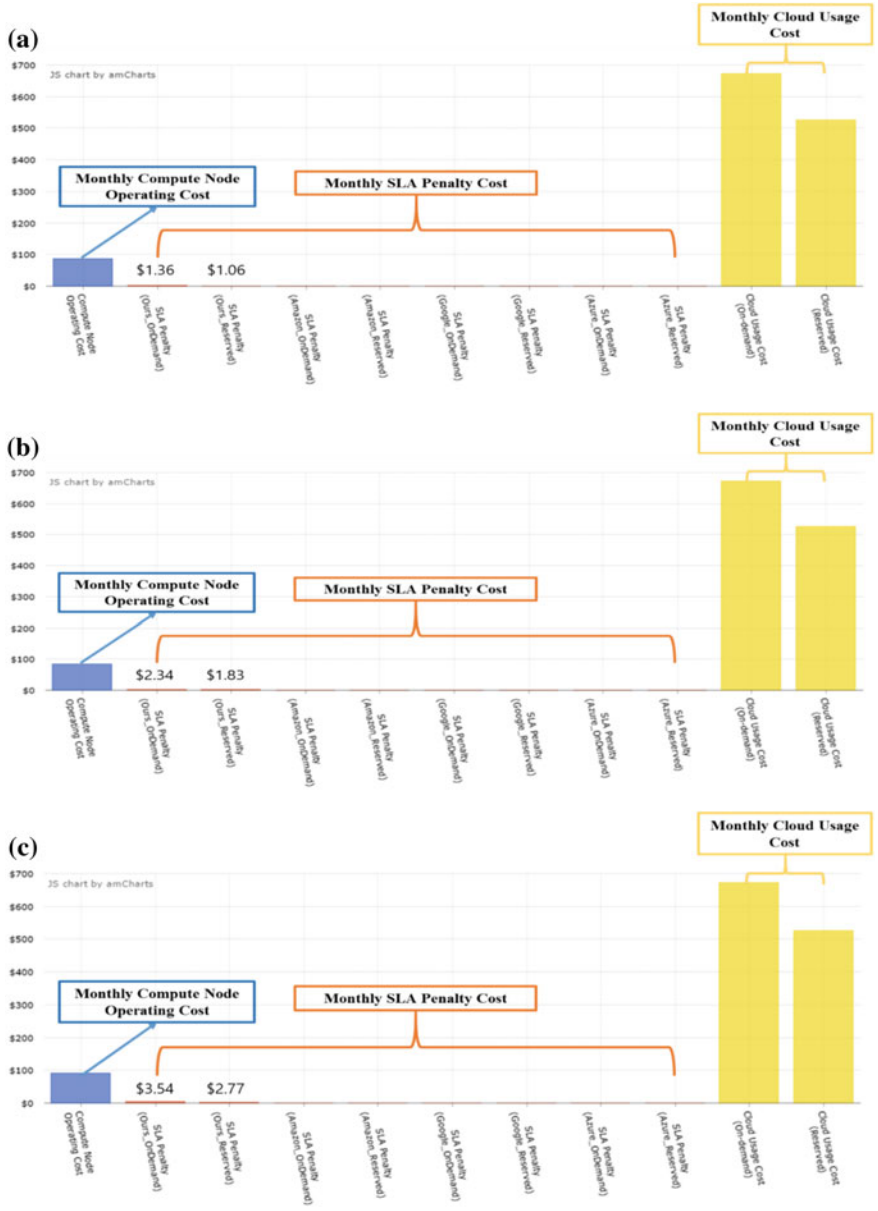


Fig. 4.19 The monthly profit analysis of cloud provider by manipulating the number of vehicles as **a** 40, **b** 60, and **c** 80

power consumption. However, we can see that the operating cost in Fig. 4.19b is smaller than that in Fig. 4.19a. The main rationale is that more service execution failure occurs in the former case, and it cause smaller VM utilization.

4.4 Conclusion

In this chapter, we handled the cloud broker system for V2C connected car services with an integrated simulation framework. The whole discussion was described in view of service offloading to a cloud. The service offloading enables to overcome the limitation of the service execution in a single vehicle and gives the four possible advantages: providing massive computing power and storage in the cloud, cost saving by reducing the computation in the vehicle, reducing power consumption by in-vehicle execution, and enabling to provide cloud-centric services. We firstly classified the V2C services into three categories: LDM, infotainment, and driving assistance. We not only introduced the examples of the V2C services belonging to each category, but also analyzed their offloadability. We then presented the execution environment for the V2C services and discussed the architecture of the cloud broker system to offload the V2C services with various offloading strategies. In the discussion, strategies like network-adaptive offloading, deadline-based offloading, and cost-effective deadline-based offloading were considered. For the evaluation of the service offloading, we introduced the integrated road traffic-network-cloud simulation framework. The framework enables the simulation users to evaluate the end-to-end performance and cost in the user-specified V2C environment. We also a way to improve the accuracy of the integration simulation via the estimation of the unknown simulation parameters. We finally showed a proof-of-concept study of the service execution with the cloud broker system using the simulation framework, and the service feasibility was analyzed in terms of performance and cost.

References

1. SBD, Connected Car Global Forecast 2015 (2015)
2. H. Shimada, A. Yamaguchi, H. Takada, K. Sato, Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems. J. Transp. Technol. **5** (2015)
3. CARASSO (Online), https://aws.amazon.com/solutions/case-studies/bmw
4. Compass4D (Online), http://www.compass4d.eu
5. Drivenet (Online), http://drivenet.pilotlab.co
6. SAFESPOT (Online), www.safespot-eu.org
7. Blue Link (Online), http://bluelink.hyundai.com
8. iDrive (Online), http://www.bmw.com/com/en/insights/technology/technology_guide/articles/idrive.html
9. Uconnect (Online), http://www.driveuconnect.com
10. Android Auto (Online), https://www.android.com/auto
11. CarPlay (Online), http://www.apple.com/ios/carplay
12. Dragon Drive (Online), http://www.nuance.com/for-business/mobile-solutions/dragon-drive/index.htm
13. Sirius (Online), http://sirius.clarity-lab.org/
14. S. Kumar, S. Gollakota, D. Katabi, A cloud-assisted design for autonomous driving, in *Proceedings of MCC'12*

15. A. Ashok, P. Steenkiste, F. Bai, Enabling vehicular applications using cloud services through adaptive computation offloading, in *Proceedings of MCS'12*
16. Pivotal (Online), https://pivotal.io
17. Blind Motion (Online), https://blindmotion.github.io/2015/04/11/ml-in-navigation
18. K. Kumar, Y.H. Lu, Cloud computing for mobile users: can offloading computation save energy? Computer **43**(4) (2010)
19. E. Lagerspetz, S. Tarkoma, Mobile search and the cloud: the benefits of offloading, in *Proceedings of PERCOM Workship* (2011)
20. S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, ThinkAir: dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in *Proceedings of INFOCOM'12*
21. A. Saarinen, M. Siekkinen, Y. Xiao, J.K. Nurminen, M. Kemppainen, P. Hui, SmartDiet: offloading popular apps to save energy, in *Proceedings of SIGCOMM'12*
22. S. Simanta, G.A. Lewis, E. Morris, K. Ha, M. Satyanarayanan, A reference architecture for mobile code offload in hostile environments, in *Proceedings of WICSA'12*
23. M.V. Barbera, S. Kosta, A. Mei, J. Stefa, To offload or not to offload? The bandwidth and energy costs of mobile cloud computing, in *Proceedings of INFOCOM'13*
24. Y. Nimmagadda, K. Kumar, Y.H. Lu, C.S.G. Lee, Real-time moving object recognition and tracking using computation offloading, in *Proceedings of IROS'10*
25. M.Y. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, R. Govindan, Odessa: enabling interactive perception applications on mobile devices, in *Proceedings of MobiSys'11*
26. Y. Zhang, H. Liu, L. Jiao, X. Fu, To offload or not to offload: an efficient code partition algorithm for mobile cloud computing, in *Proceedings of ClOUDNET'12*
27. H. Flores, S. Srirama, Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning, in *Proceedings of MCS'13*
28. F. Xia, F. Ding, J. Li, X. Kong, L.T. Yang, J. Ma, Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. Inf. Syst. Front. **16** (1) (2014)
29. P. Cooper, U. Dolinsky, A.F. Donaldson, A. Richards, C. Riley, G. Russell, Offload—automating code migration to heterogeneous multicore systems, in *Proceedings of HiPEAC'10*
30. H.Y. Chen, Y.H. Lin, C.M. Cheng, COCA: computation offload to clouds using AOP, in *Proceedings of CCGrid'12*
31. M.S. Gordon, D.A. Jamshidi, S. Mahlke, Z. M. Mao, X. Chen, COMET: Code Offload by Migrating Execution Transparently, in *Proceedings of OSDI'12*
32. D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing. IEEE Trans. Wireless Commun. **11**(6) (2012)
33. E. Cuervo, A. Balasubramanian, D.K. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in *Proceedings of MobiSys'10*
34. B.G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in *Proceedings of EuroSys'11*
35. D. Kovachev, T. Yu, R. Klamma, Adaptive computation offloading from mobile devices into the cloud, in *Proceedings of ISPA'12*
36. Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, S. Yang, Refactoring android java code for on-demand computation offloading, in *Proceedings of OOPSLA'12*
37. H. Wu, Q. Wang, K. Wolter, Tradeoff between performance improvement and energy saving in mobile cloud offloading systems, in *Proceedings of ICC'13*
38. C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, E. Zegura, COSMOS: computation offloading as a service for mobile devices, in *Proceedings of MobiHoc'14*
39. B. Zhou et al., A context sensitive offloading scheme for mobile cloud computing service, in *Proceedings of CLOUD'15*
40. H. Kim, J. Han, S.-H. Kim, J. Choi, D. Yoon, M. Jeon, E. Yang, N. Pham, S. Woo, D. Kim, C.-H. Youn, IsV2C: an integrated road traffic-network-cloud simulator for V2C connected car services, *submitted to SCC'17*

41. OpenStreetMap (Online), http://www.openstreetmap.org/
42. SUMO (Online), http://www.sumo.dlr.de/
43. ns-3 (Online), http://www.nsnam.org/
44. Lena (Online), http://networks.cttc.es/mobile-networks/software-tools/lena/
45. Amazon EC2 (Online), htttp://aws.amazon.com/ec2
46. Google Compute Engine (Online), http://cloud.google.com
47. Microsoft Azure (Online), http://azure.microsoft.com
48. J. Hamilton, Cost of power in large-scale data centers, *Keynote, at ACM SIGMETRICS 2009* (Online), http://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers
49. Universal Mobile Telecommunications System (UMTS); Packet Data Convergence Protocol (PDCP) specification, Technical Specification, ETSI TS 125 323 V5.0.0 (2002)
50. X. Zhang, Z. Hu, X. Du, Probabilistic inverse simulation and its application in vehicle accident reconstruction. J. Mech. Des. **135**(12) (2013)
51. T. Flessa, E. McGookin, D. Thomson, Numerical stability of inverse simulation algorithms applied to planetary rover navigation, in *Proceedings of MED'16*
52. Y. Liu, J. Jiang, Inverse dynamics of vehicle minimum time manoeuvre for collision avoidance problem. Int. J. Vehicle Saf. **9**(2) (2016)
53. S.I. You, J.Y.J. Chow, S.G. Ritchie, Inverse vehicle routing for activity-based urban freight forecast modeling and city logistics. Transp. A Transport Sci. **12**(7) (2016)
54. L. Zha, D. Lord, Y. Zou, The Poisson inverse Gaussian (PIG) generalized linear regression model for analyzing motor vehicle crash data. J. Transp. Saf. Secur. **8**(1) (2016)
55. OpenStack (Online), http://www.openstack.org/
56. Naver MAPS API (Online), http://navermaps.github.io/maps.js/
57. Cisco, Cisco visual networking index: forecast and methodology (2015) (Online), http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html
58. C. Jennings, A. Narayanan, D. Burnett, A. Bergkvist, WebRTC 1.0: Real-time communication between browsers, *W3C, W3C Ed. Draft. Aug*, no. May, 2014
59. M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, The Secure Real-time Transport Protocol (SRTP). *Internet Soc. RFC 3711* **1**, 1–56 (2004)
60. F. Wang, J. Liu, M. Chen, CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities, in *Proceedings of INFOCOM*'12
61. A. Amirante, T. Castaldi, L. Miniero, S.P. Romano, Janus: a general purpose WebRTC gateway, in *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications*, (2014), pp. 7:1–7:8
62. W.-J. Kim, H. Jang, G.-B. Choi, I.-S. Hwang, C.-H. Youn, A WebRTC based live streaming service platform with dynamic resource provisioning in cloud, in *Proceedings of TENCON'16*
63. Docker (Online), https://www.docker.com
64. RabbitMQ (Online), https://www.rabbitmq.com
65. Docker-Swarm (Online), https://www.docker.com/products/docker-swarm
66. cAdvisor (Online), https://github.com/google/cadvisor
67. influxDB (Online), https://influxdata.com/
68. E. Diaconescu, The use of NARX neural networks to predict chaotic time series. WSEAS Trans. Comput. Res. **3**(3), 182–191 (2008)
69. S. Dutta, T. Taleb, A. Ksentini, QoE-aware elasticity support in cloud-native 5G systems, in *Proceedings of ICC'16*
70. Amazon EC2, https://aws.amazon.com/ko/directconnect
71. R. Wang, M. Xue, K. Chen, Z. Li, T. Dong, Y. Sun, BMA: Bandwidth allocation management for distributed systems under cloud gaming, in *ICCSN* (2015)
72. X. Qi, Q. Yang, D. Nguyen, G. Zhou, G. Peng, LBVC: towards low-bandwidth video chat on smartphones, in *MMSys* (2015)
73. H. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, A. Venkataramani, A structural approach to latency prediction, in *SIGCOMM* (2006)

74. Y. Wu, B. Li, L. Zhang, Z. Li, F.C.M. Lau, Scaling social media applications into geo-distributed clouds. IEEE/ACM Trans. Networking **23**(3) (2015)
75. H. Kim, J. Han, S.-H. Kim, J. Choi, D. Yoon, M. Jeon, E. Yang, N. Pham, S. Woo, D. Kim, C.-H. Youn, IsV2C: an integrated road traffic-network-cloud simulator for V2C connected car services, in *submitted to SCC'17*
76. H. Shimada, A. Yamaguchi, H. Takada, Implementation and evaluation of local dynamic map in safety driving systems. J. Transp. Technol. **5**, 102–112 (2015). (April)
77. J. Hauswald, L. Tang, J. Mars, M.A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R.G.G. Dreslinski, T. Mudge, V. Petrucci, Sirius: an open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers, in *Proceedings of ASPLOS'15*

Chapter 5
Mobile Device as Cloud Broker for Computation Offloading at Cloudlets

5.1 Introduction

5.1.1 Overview of the Cloud Category

With the development of cloud computing and the evergrowing number of mobile devices, many applications require higher user's quality of experience (QoE). Those applications include computation-intensive tasks for processing, while offloading the tasks to cloud incurs high communication cost and large latency. In order to speed up the task execution of latency-sensitive tasks, there proposed a kind of edge cloud. The edge clouds are also called as fog clouds, cloudlets, mobile clouds and mobile cloudlets. Cloudlet can be deemed as a very small data center consisting of network terminals, while mobile cloudlet is constituted through connection of mobile device to D2D. Edge cloud supports resource-intensive and interactive mobile applications by providing powerful computing resources to mobile devices with lower latency. Hence, there constitutes the framework of cloud, that is, mobile device-edge cloud-remote cloud. The composition of system infrastructure is shown in the Fig. 5.1.

5.1.2 Computation Offloading from Remote Cloud to Mobile Cloudlet

With an ever-increasing number of mobile devices and the resulting explosive mobile traffic, 5G networks call for various technology advances to transmit the traffic more effectively while changing the world by interconnecting tremendous amount of mobile devices [1]. However, mobile devices have limited communication and computation capabilities in terms of computation power, memory, storage, and energy. In addition to the broadband bandwidth support from 5G, cloud computing needs to be utilized to enable mobile devices to obtain virtually unlimited

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_5

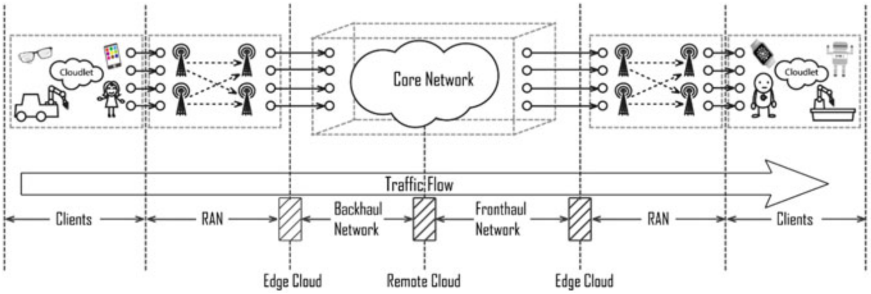
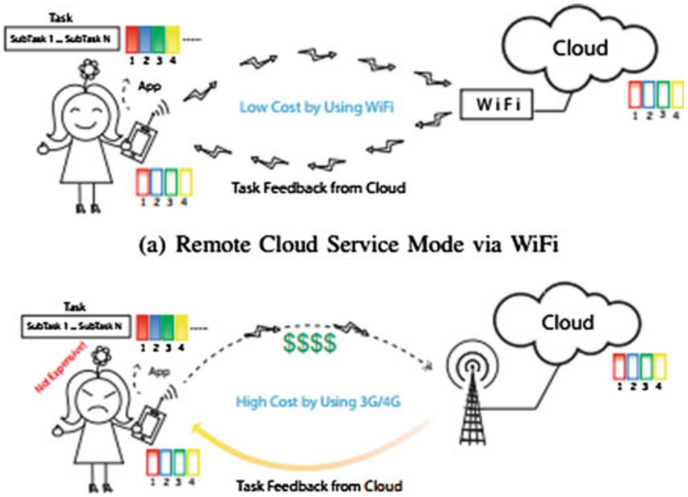


Fig. 5.1 Illustration of the category of cloud: mobile device, cloudlet, remote cloud

dynamic resources for computation, storage and service provision that will overcome the constraints in the smart mobile devices. Thus, the combination of 5G and cloud computing technology are paving the way for giving rise to more attractive applications, which involve compute-intensive task execution at "small" device carried by mobile use who enjoys "large" capabilities enabled through the new technologies.

With the support of mobile cloud computing (MCC) [2], a mobile user basically has one more option to execute the computation of its application, i.e., offloading the computation to the cloud [3]. Thus, one principal problem is that, under what conditions a mobile user should offload its computation to the cloud [4]? An illustrative scenario of computation offloading at remote cloud is shown in Fig. 5.2a, where a user is covered by WiFi. Since the terminal device at user end has limited resources, i.e., hardware, energy, bandwidth, etc., the cellphone itself is infeasible to finish some compute-intensive task. Instead, the data related to the computation task can be offloaded to the remote cloud via WiFi which is free of charge. After the execution of the computation task in the remote cloud, the computation result will be sent back to the user's cellphone. We call this computation offloading mode as "remote cloud service" (RCS), as shown in Fig. 5.2. Typically, WiFi is used in RCS and user mobility is seriously limited within areas covered by WiFi. In the case that a mobile user moves into an outdoor environment without WiFi coverage, the computation task must be offloaded to the remote cloud via cellular network, which results in high communication cost. Compared with traffic offloading [5, 6], applications involving computation offloading usually are more delay-sensitive. Thus, how to design a cost-effective computation offloading service mode while achieving good Quality of Experience (QoE) becomes an important problem to be solved in this work.

With the development of mobile devices in terms of increased memory and computational capability, a novel peer-to-peer communication model for MCC is proposed to interconnect nearby mobile devices through various short range radio communication technologies (e.g., WiFi, Bluetooth, etc.) to form mobile cloudlets, where a mobile device can work as either a computational service provider or a client of service requester [7]. In recent years, direct short range communications between cellular devices known as device-to-device communications (D2D) have



(a) Remote Cloud Service Mode via WiFi

(b) Remote Cloud Service Mode via 3G/4G

Fig. 5.2 Illustration of computation offloading through remote cloud service mode

been widely studied, where a comprehensive discussion of related usage cases and business models are given in [8]. In [9], a fundamental question while utilizing cloudlet is proposed as follows: whether and under what conditions mobile cloudlet is feasible for providing mobile application services. To answer the question, cloudlet size, cloudlet node's lifetime and reachable time are defined. However, the computation offloading is limited only when a computation node (i.e., cloudlet service requester) keeps contact with a service node (i.e., cloudlet service provider) [9]. We denote the previous computation offloading mode for cloudlet assisted service as "connected ad hoc cloudlet service" (CCS). Under CCS mode, the computation task is offloaded to another local cloudlet node when there are available D2D connectivities during the whole procedure including computation offloading, computation execution, and computation feedback. Though local D2D connectivity might be unavailable due to network dynamic, one advantage of employing CCS is the lower communication cost and short transmission delay compared to the case when the computation task is offloaded to remote cloud. We articulate the features of RCS and CCS mode as follows:

- **RCS mode**: with the support of stable 3G/4G, computation nodes offload their computation tasks to remote cloud at any time. The advantages of remote cloud mode include high reliability and the provisioning of on-demand service. The disadvantage is the higher cost by using cellular network resource.
- **CCS mode:** once available D2D connectivity is built up between computation node and service node, the energy cost is economical since local wireless (e.g., WiFi) can be utilized for content delivery and whatever communications involved to lead the completion of the offloaded computational task. The

limitation of CCS mode is the strict requirement on the contact duration between a computation node and a service node to guarantee enough processing time for the offloaded computational task. Once a computation node and a service node are disconnected due to mobility or other network dynamics while the offloaded computational task is not finished, the computation execution is failed.

5.1.3 Cloud Broker from Cloud to Mobile Device

According to discussions in previous sections, cloud broker used by remote cloud is capable of arranging cloud resources and user tasks overall and dispatching work flows and virtual machines, so as to reduce the latency and energy consumption to the maximum extent. In case of computation offloading from remote cloud to mobile cloudlet, our cloud broker also is transferred from the remote to mobile device. That is to say, in case of the utilization of mobile cloudlet, cloud broker of the moment becomes the mobile device with tasks, and it is capable of perceiving user tasks and surrounding resources, and knows how to dispatch those tasks and resources reasonably, so as to attain the minimum latency and energy consumption of task processing. Here, we call mobile device with the functions of cloud broker as computation node. And mobile cloudlet composed of mobile device and processing tasks is called service node. Computation task requiring for processing is called computation task.

- A computation task can be divided into multiple sub-tasks.
- Depending on specific applications, the sub-tasks are different with each other or identical.
- The content delivery for a sub-task can be finished during a short contact period between computation node and service node, while the computation execution incurs relatively longer delay.
- Each service node only can be utilized for one sub-task.
- We don't consider packet loss and the communication cost is decided by the amount of data transferred in the network.

Table 5.1 shows the detailed terminologies.

Table 5.1 Definition of Terminologies

Terminology	Definition
Computation task	Data associated with a computation
Computation node	A node which has a computation task to be executed, it can also be called Task node
Service node	A node which is available to provide service for computation node to handle a sub-task
Sub-task	Multiple sub-tasks consist of a computation task
Sub-task result	The execution result of a sub-task by service node

5.2 New Architecture of Computation Offloading at Cloudlet

The major focus of previous work is to minimize the cost by finding an optimal tradeoff between CCS mode and RCS mode. We propose a novel service mode for cloudlet-assisted computing by considering the following realistic scenario. The typical contact duration might be too short to guarantee a valid computation offloading, execution and result feedback under CCS mode. However, it is reasonable to presume that the contact duration is enough to transmit the content associated with the computation to the service node via D2D connectivity. After the connection between the computation node and the service node is over, the computation is still processed in the service node for a certain amount of time until the sub-task execution is finished. We call this new service mode as "opportunistic ad hoc cloudlet service" (OCS). The basis idea of OCS is the utilization of the opportunistic contacts among computation node and service nodes while not limiting the mobility of the user. It is assumed that each computation task has a certain deadline, by the end of which the computation result should be sent back from service nodes to computation node, and the locations of the service nodes have three possibilities: (i) meeting computation node again; (ii) losing D2D connectivity with computation node while seeking help from 3G/4G; (iii) losing connectivity with computation node while WiFi is available. Based on the above three scenarios, we divide OCS service modes into three categories as follows:

- **OCS (back & forth)**: In [10], Li et al. proposed a mobility-assisted computation offloading scheme, which calculates the probability of meeting twice between a pair of computation node and service node. To calculate the probability, the statistics of node mobility are used. Before the computation task deadline, once service node meets computation node again while the execution of the allocated sub-task is finished, the sub-task result can be successfully sent to the computation node. We call this computation offloading service mode via ad hoc cloudlet as "back-and-forth service in cloudlet". However, the user mobility under this mode is typically limited within a certain area, in order to guarantee the second meeting between the computation node and the service node. The mobility support of OCS (back&forth) mode should be higher than that of CCS and RCS (WiFi). Thus, the rank of mobility support is marked as "medium" in Table 5.2.
- **OCS (one way-3G/4G)**: It's challenging to achieve cost-effective computation offloading without sacrificing the mobility support and the mobile nodes' freedom, i.e., a service node might roam to another cell. For sake of generality, let's consider the scenario without WiFi coverage, where service node needs to upload the sub-task result to cloud via 3G/4G. Typically, the data size of sub-task result (S_{sub-tk}^{result}) is smaller than the size of the original sub-task that service node receives (i.e., S_{sub-tk}^{recv}). Let r denote the ratio of S_{sub-tk}^{result} and S_{sub-tk}^{recv}. The lower r is, the better performance of OCS (one way-3G/4G) mode will be.

Table 5.2 A comparison of service modes for task offloading

Structure	Service Mode	Cost	Scalability	Mobility Support	Freedom of Service Node	Computation Duration
Remote Cloud	RCS (3G/4G)	High	Coarse	High	N/A	Medium
	RCS (WiFi)	Low	Coarse	Low	N/A	Medium
Ad Hoc Cloudlet	CCS	Low	Coarse	Low	Low	Low
	OCS (back \&forth)	Low	Medium	Medium	Medium	High
	OCS (one way-3G/4G)	Medium	Fine	High	High	High
	OCS (one way-WiFi)	Low	Fine	High	High	High

- **OCS (one way-WiFi)**: In case that the service node roams to a different cell which is covered by WiFi, e.g., the mobile user goes back to home, the sub-task result can be uploaded to the cloud via WiFi. For most practical values of *r*, the communication cost under this service mode is between RCS (WiFi) and RCS (3G/4G).

Table 5.2 compares the features of various computation offloading service modes in terms of different performance metrics, such as cost, scalability, mobility support, freedom of service node, and computation duration.

Figure 5.3 shows illustrative examples to explain the above three OCS service modes. *Rachel* gets a compute-intensive task, which is infeasible to be executed

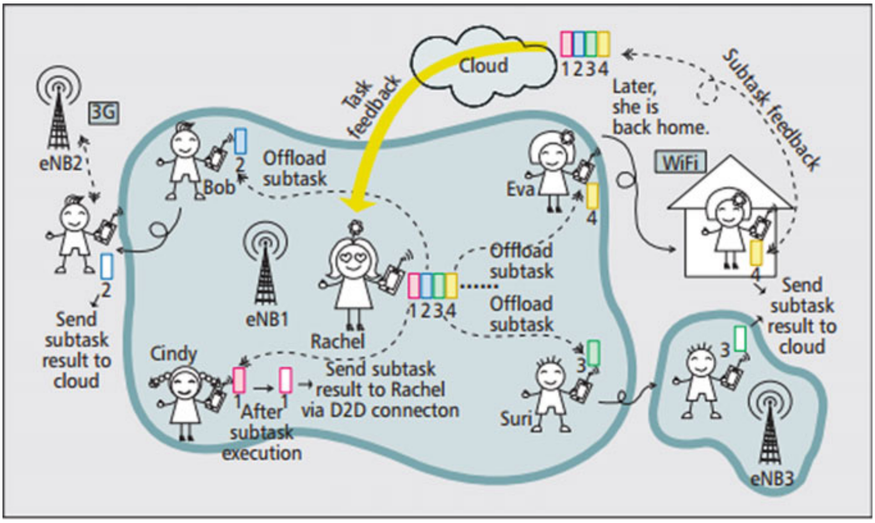


Fig. 5.3 Illustration of the computation offloading at Ad Hoc Cloudlet

timely by her own mobile phone. Within the range of D2D connectivity, *Rachel* has four friends named *Bob*, *Eva*, *Cindy* and *Suri* whose mobile phones are in idle status. Thus, *Rachel* divides the computation task into four sub-tasks, and forwards the corresponding contents to their four mobile phones via D2D links, respectively. *Cindy* does not move much, and keeps connectivity with *Rachel*. After execution, *Cindy*'s sub-task result is sent to *Rachel* directly under either CCS or OCS (back&forth) service mode. In comparison, *Bob* and *Suri* moves to another cell before the end of sub-task execution. Thus, they use OCS (one way- 3G/4G) service node to upload sub-task result to the cloud. As for *Eva*, let's assume she comes back to her home with the WiFi support, and thus utilizing OCS (one way-WiFi) service mode.

Since OCS does not require that both computation node and service nodes should keep contact or locate in a certain area, it has higher scalability. In fact, OCS is especially useful in some applications where the size of data content associated with a task is large while the size of result data is relatively small. Given application of image segmentation as example, the size of pictures generated by an image sensor of the mobile device can be large. Thus transmitting the whole picture to the cloud via 3G/4G link consumes much valuable bandwidth and energy. Under OCS (one way-3G/4G) mode, computation node first delivers the whole picture along with image segmentation code to a service node via D2D links. After performing the segmentation code for the whole picture, only a certain region of interest (ROI) in the picture is obtained as the computation result. Uploading the small size of ROI to the cloud via 3G/4G links leads to cost saving compared to RCS mode, while OCS offers more freedom for computation node and service node without the requirement of a strict contact duration compared to CCS mode. Moreover, in some hostile environments, for example military operations and disaster recovery, RCS mode is usually not viable in such emergent situations. By comparison, OCS mode is more flexible to overcome this problem. Therefore, OCS mode can be considered as a novel compromised mode between CCS mode and RCS mode, and thus yielding more flexibility and cost effectiveness to enable a more energy-efficient and intelligent strategy for computation offloading by the use of cloudlet.

5.3 A Study on the OCS Mode

5.3.1 Computation Allocation

Before offloading the computation task to the ad hoc cloudlet, the principal problem for the computation node is how to divide the computation task into a certain number of sub-tasks. This problem is related to the number of service nodes (N_{sn}), as well as their processing capability. Intuitively, the larger N_{sn} is, the higher accumulated capacity of computation that the service nodes possess, and the computation duration will be shorter. However, it is critical to decide an optimal number of sub-tasks, and the following strategies can be considered:

- If we assume that the number of nodes within the same cell is stable everyday, the node number can be estimated by the number counted in the last day. The number of sub-tasks can be the same as the number of service nodes.
- Based on social network, the relationships among users can be extracted. Usually, the service nodes who have strong connections with the computation node are more likely to accept the sub-task assignments.

Once we know the number of sub-tasks, the left problem is to decide how to divide the computation tasks. There are several ways to divide the computation tasks:

Static Allocation: For general case, the computation node does not know the computational capability of the service nodes. Thus, for the sake of simplicity, we assume the capability of all the service nodes is similar. The computation node assigns sub-tasks equally among service nodes. However, the weak point of this strategy is that the computation duration is calculated based on the worst delay for a sub-task execution.

Dynamic Allocation: More realistically, the capability of the service nodes should be different. When the information of service nodes in terms of processing capability can be obtained, we can achieve the assignment of sub-tasks with more intelligence. Typically, if the capability is higher, sub-task with heavier computation load will be allocated.

5.3.2 Computation Classification

For computation classification, we further divide the sub-task into two situations, i.e., (i) the sub-tasks are different from each other; (ii) the sub-tasks can be cloned to each other:

- ***Opportunistic* ad hoc *cloudlet service without computation task clone***: Given an example scenario as shown in Fig. 5.4a, Rachel has 10 pictures, each of which contains special region of interest (ROI). If the dynamic method is used, the computation node (corresponding to Rachel) can divide the computation task (i.e., handling the 10 pictures) into four various sub-tasks which are assigned to Cindy, Bob, Suri and Eva, respectively. Each person has specific computational capability. As an example, Cindy and Bob get three and two pictures, while Suri and Eva are allocated 1 and 4 pictures, respectively. Obviously, the four sub-tasks are different at two aspects. First, the number of pictures that each sub-task contains is different. Second, each picture is different. After ROIs are segmented, the result will be sent back to the computation node.

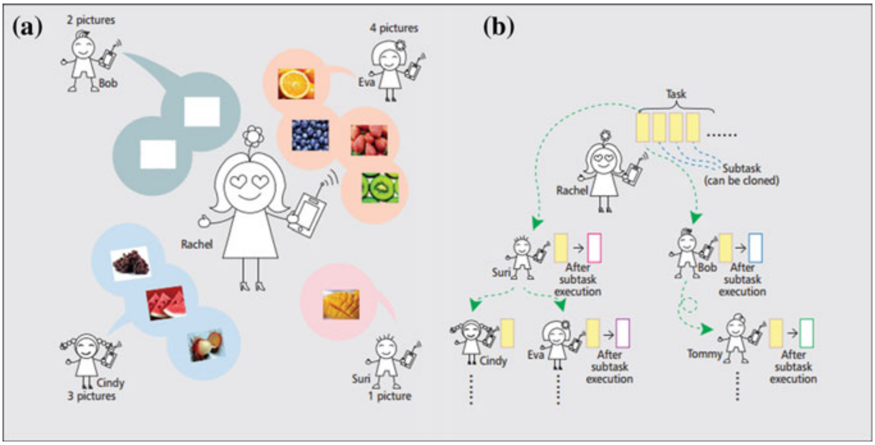


Fig. 5.4 Illustration of sub-task offloading in opportunistic ad hoc cloudlet service: **a** without computation task clone; **b** with computation task clone

- ***Opportunistic* ad hoc *cloudlet service with computation task clone***: In some application, a computation task can be divided into numerical equivalent sub-tasks. For this case, after a service node receives a sub-task, the sub-task content can be cloned and offloaded to another service node, which is similar with epidemic model. As shown in Fig. 5.4b, the sub-task containing the same content is duplicated to nearby service nodes, in order to accelerate the execution of the computation task. Regarding energy consumption aspect, the cost will be decreased since D2D is utilized during the flooding of the sub-tasks. There are two important parameters for building the model in this situation: (1) the initial number of task nodes; (2) the current number of service nodes that have the sub-tasks.

5.4 Allocation Problem in Mobile Device Broker

As shown in Table 5.2, different service modes have both advantages and disadvantages. Trade-offs are arising when we need to provide users with high QoE, while saving the communication cost and maintaining a degree of scalability for enabling a wide range of intelligent applications.

In this section, cost under these different service modes will be analyzed. Let us consider the scenario where M nodes exist in the cell and WiFi is not available by default (if the situation has WiFi, we use WiFi first). For the sake of simplicity, assume there is only one computation node, which has a computation task with a total size of computation load Q. The task can be divided into n sub-tasks. Assume each service node can process x_i in dynamic allocation, then $\sum_{i=1}^{n} x_i = Q$. Energy cost includes three parts, i.e., computation offloading, computation execution and computation feedback.

- **RCS**: let $E^{cell}_{n\rightarrow c}$ denote the per unit communication cost from computation node to cloud; let $E^{cell}_{c\rightarrow n}$ denote the per unit communication cost for cloud-based result feedback; let E^{cloud}_{proc} denote the per unit energy cost for computation tasks processed in cloud. Then, the total cost at cloud can be calculated as:

$$\begin{aligned} C_{RCS} &= \sum_{i=1}^{n} (E^{cell}_{n\rightarrow c} x_i + E^{cloud}_{proc} x_i + rE^{cell}_{c\rightarrow n} x_i) \\ &= Q(E^{cell}_{n\rightarrow c} + E^{cloud}_{proc} + rE^{cell}_{c\rightarrow n}) \end{aligned} \tag{5.1}$$

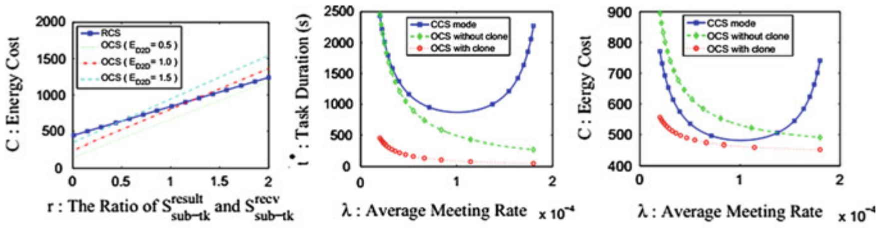
- **CCS**: The major energy consumption is caused by D2D communications and energy of periodically probing the surrounding nodes. Let E_{D2D} denote the per unit communication cost from computation node to service node; let E^{node}_{proc} denote the per unit energy cost for service node to process a sub-task locally; let ρ denote the probing cost per time unit; let t^* denote the task duration for a successful computation offloading. This is related to the average meeting rate of two nodes in the cell, we denote it as λ. Then,

$$\begin{aligned} C_{CCS} &= \sum_{i=1}^{n} (E_{D2D} x_i + E^{node}_{proc} x_i + rE_{D2D} x_i) + M\rho t^* \\ &= Q(E_{D2D} + E^{node}_{proc} + rE_{D2D}) + M\rho t^* \end{aligned} \tag{5.2}$$

- **OCS**: If we consider a typical scenario where WiFi is not available when a service node roams to another cell, OCS will include two cases, i.e., OCS (back&forth) and OCS (one-way-3G/4G). In case of OCS (back&forth), we need to consider the probability (P, $0 \leq P \leq 1$) of service node meeting computation node twice, where D2D is utilized to deliver the sub-task result. Otherwise, cellular network is the only way for communications due to the intrinsic feature of OCS (one-way-3G/4G). We use the same symbols as above, Then,

$$\begin{aligned} &C_{OCS} = Q(E_{D2D} + E^{node}_{proc}) + rPQE_{D2D} + r(1-P) \\ &Q(E^{cell}_{n\rightarrow c} + E^{cell}_{c\rightarrow n}) + M\rho t^* \end{aligned} \tag{5.3}$$

Typically, the cost for a service node to offload computation task to the cloud or for the cloud to sent back the computation result to the computation node via 3G/4G are larger than D2D cost, i.e., $E^{cell}_{n\rightarrow c}, E^{cell}_{c\rightarrow n} > E_{D2D}$; the processing cost in the cloud is large than in service node, i.e., $E^{cloud}_{proc} > E^{node}_{proc}$. Thus, considering energy and delay under various scenarios, it is expected that flexible trade-offs should be achieved according to specific application requirements. Figure 5.5a shows the



(a) Comparison of energy cost between RCS and OCS with various r

(b) Comparison of energy cost between CCS and OCS with various λ

(c) Comparison of task duration between CCS and OCS with various λ

Fig. 5.5 Comparison of energy cost between RCS and OCS with various r

comparison of cost in terms of RCS and OCS mode. The cost of RCS is represented by the solid blue curve, while the other lines represent the cost of offloading by the use of OCS mode with various r. As $E^{cell}_{n\to c}, E^{cell}_{c\to n} > E_{D2D}$, so when r is less than 1, OCS mode with low E_{D2D} always has lower cost than RCS. However, when r is larger than 1, the cost of OCS increases with the increase of r and growth speed is larger than the growth rate of RCS. Furthermore, When E_{D2D} increases, the cost of OCS becomes larger. In summary, OCS outperforms RCS in following three cases: (1) $E_{D2D} = 0.5$, $r < 2$; (2) $E_{D2D} = 1$, $r < 1.15$; (3) $E_{D2D} = 1.5$, $r < 0.5$, as shown in Fig. 4a.

In Fig. 5.5b, the cost of CCS and OCS is compared. Among the three schemes, OCS with computation clone exhibits the lowest cost and always outperforms than the other schemes. This is because OCS with clone yields the fastest speed to complete the task. When $0.00002 \le \lambda \le 0.00014$, CCS outperforms OCS without computation clone in terms of energy cost, since OCS without computation clone needs to upload sub-task results to the cloud while CCS saves this cost. When λ increases, the contact duration becomes smaller which may cause the failure of sub-task's execution in CCS. Thus, when λ is larger than 0.00014$, OCS without computation clone has better performance than CCS.

Figure 5.5c shows the comparison of computation duration in terms of OCS and CCS modes. Given a fixed λ, computation duration of OCS is shorter than that of CCS. With the increase of λ, OCS yields better delay performance. It is because the frequency of computation node meeting with service nodes increases with a larger λ. For CCS, computation duration gradually decreases from a small λ (e.g., 0.00002–0.0001). However, with the continuous increase of λ, the computation duration of CCS starts to increase again. It is because the contact duration (meeting time) becomes smaller, which causes the insufficient contact time to enable a successful sub-task offloading, execution and feedback. As discussed above, we can draw the conclusion that:

- **RCS mode**: If the computation node is highly sensitive to delay, i.e., the user can afford higher cost to achieve good QoE, 3G/4G can be used to enable computation execution at RCS.

- **CCS mode**: If the computation node has major concern in terms of cost while the movement of service node is limited, CCS is a good choice.
- **OCS mode**: If the size of the computation result is much smaller than the size of computation task, i.e., $r$ is lower, OCS is more cost-effective while enabling maximum freedom for computation node and service nodes.

References

1. V. Leung, T. Taleb, M. Chen, T. Magedanz, L. Wang, R. Tafazolli, Unveiling 5G wireless networks: emerging research advances, prospects, and challenges. IEEE Network **28**(6), 3–5 (2014)
2. T. Taleb, A. Ksentini, Follow me cloud: interworking federated clouds and distributed mobile networks. IEEE Network **27**(5), 12–19 (2013)
3. H. Flores, S. Srirama, Mobile Code Offloading: Should It Be A Local Decision Or Global Inference? in *Proceeding of ACM MobiSys*, (2013)
4. M. V. Barbera, S. Kosta, A. Mei, J. Stefa, To Offload or Not to offload? The Bandwidth and Energy Costs for Mobile Cloud Computing, in *Proceedings of IEEE INFOCOM*, (2013)
5. B. Han, P. Hui, V.S.A. Kumar, M.V. Marathe, J. Shao, A. Srinivasan, Mobile data offloading through opportunistic communications and social participation. IEEE Trans. Mob. Comput. **11**(5), 821–834 (2012)
6. X. Wang, M. Chen, Z. Han, et al., TOSS: Traffic Offloading By Social Network Service-Based Opportunistic Sharing In Mobile Social Networks, in *Proceedings of IEEE INFOCOM*, (2014)
7. H. T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Commun. Mob. Comput. (2011)
8. L. Lei, Z. Zhong, C. Lin, S. Shen, Operator controlled device-to-device communications in LTE-advanced networks. IEEE Wireless Commun. **19**(3), 96–104 (2012)
9. Y. Li, W. Wang, Can Mobile Cloudlets Support Mobile Applications? in *Proceedings of IEEE INFOCOM*, (2014)
10. C. Wang, Y. Li, D. Jin, Mobility-Assisted Opportunistic Computation Offloading. IEEE Commun. Lett. **18**(10), 2014

Chapter 6
Opportunistic Task Scheduling Over Co-located Clouds

6.1 Introduction

Nowadays, due to the explosive increase of mobile devices and data traffic, various innovative technologies have been developed to transfer data more efficiently by the use of large quantities of mobile devices connected with each other. However, as mobile devices have limitations in terms of computing power, memory, storage, communications and battery capacity, the computation-intensive tasks are hard to be handled locally. Fortunately, the paradigm of mobile cloud computing (MCC) enables mobile devices to obtain extra resources for computing, storage and service supply, and may overcome above limitations [1]. Typically, computation-intensive tasks can be uploaded to the remote cloud [2] through cellular network or WiFi. Though WiFi is energy efficient with high data rate, its connections are intermittent in mobile environments. In contrast to WiFi, cellular network provides stable and ubiquitous connections with high cost.

In recent years, as a direct short-range communication mode between devices in the same district, device-to-device communication (D2D) has been well studied in terms of its techniques, application cases, and business models [3–5]. With the enormous increase of mobile devices with high memory and computing power, a new type peer-to-peer communication mode for MCC, called ad hoc cloudlet or mobile cloudlets, has been introduced [6]. In a mobile cloudlet, a mobile device can be either a service node or a computing service requester (referred to as task node). When the connection of D2D is available in the mobile cloudlets, task node can offload the computing task to the cloudlet. The use of mobile cloudlets leads to low communication costs and short transmission delay, however, the intermittent D2D connections may quickly become invalid due to network dynamics.

In mobile environment, remote cloud and mobile cloudlets both have advantages and disadvantages for task offloading. The keypoint of our work in this chapter is to find the compromised service mode by the use of remote cloud and mobile cloudlets to minimize the cost and still ensure good-enough quality of experience

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_6

Table 6.1 A comparison of service modes for task offloading

Structure	Communication style	Cost	Scalability	Mobility support	Freedom of service node	Computation duration
Remote cloud	Cellular network	High	Coarse	High	N/A	Medium
	WiFi	Low	Coarse	Low	N/A	Medium
Mobile cloudlets	D2D	Low	Coarse	Low	Low	Low
Co-located clouds	D2D	Low	Medium	Medium	Medium	High
	D2D and cellular network	Medium	Fine	High	High	High
	D2D and WiFi	Low	Fine	High	High	High

(QoE) [7]. As shown in Table 6.1, remote cloud based service mode has shortcoming of high cost, while the efficiency of mobile cloudlets oriented service mode is closely related with user's mobility. To solve the problem, in this chapter, we proposes a new task offloading mode named "Opportunistic task Scheduling over Co-located Clouds" (OSCC) which divides into three categories including OSCC (back\&forth), OSCC (one way-WiFi) and OSCC (one way-Cellular Network). We analyze the performance of the OSCC mode extensively in terms of task duration and energy cost under different application scenarios. In this chapter, the energy cost mainly includes communication cost and processing cost. The communication cost is consumed for task offloading, computation result feedback. The communications can be achieved by either D2D link or cellular network. The processing cost consists of processing energy cost in either clouds or local nodes. The optimal solution of task allocation is given to achieve minimum energy cost. Basically, it's more energy efficient to allocate more workloads to the service nodes with higher mobility and larger computing capacities. Furthermore, we introduce two different kinds of task allocation schemes, i.e., dynamic allocation and static allocation. Under both mobile cloudlets mode and OSCC mode, dynamic allocation exhibits lower cost than static allocation. We provide some insights based on the performance evaluation, and the following question is answered: given a computation task, what is the optimal task partitioning strategy, i.e., how many sub-tasks should be divided to optimize the integrated performance in terms of task duration and energy cost. In conclusion, the OSCC mode outperforms remote cloud mode and mobile cloudlets mode due to a better tradeoff between cost and mobility support.

6.2 Background and Related Works

We introduce the existing methods for task offloading, which are classified into two categories: (1) based on the remote cloud, (2) with the help of mobile cloudlets.

6.2.1 *Task Offloading Based on Remote Cloud*

Along with the development of MCC, mobile users can upload their computing tasks [8, 9] to the cloud and the cloud will return the result to them after the completion of the computing tasks [10, 11]. This is traditional task offloading mode as shown in Fig. 6.1. Mobile devices can offload computing related tasks to the cloud in two ways. One is through WiFi for cost saving as shown in Fig. 6.1a while the other is through expensive cellular network (e.g. 3G/4G/5G) as shown in Fig. 6.1b in case WiFi is unavailable. Therefore, a major question is: under what situation should the mobile users offload the computing tasks to the cloud [12]? Previous work introduced various offloading strategies. Clonecloud [13] has proposed cloud-augmented execution by using cloned virtual image as a powerful virtual unit. Kosta et al. [14] has proposed a dynamic resource allocation and the framework of parallel execution named ThinkAir. As for the parallel task [15] allocation on the mobile devices, Jia et al. [16] designed a kind of heuristic offloading scheme. Different from the existed research work, Lei et al. [17] first considers the interactions between the offloading decision function of MCC and the radio resource management function of wireless heterogeneous network (HetNet), and the offloading decision is made considering both the offloading gain and the cost of using the HetNet when a Service Level Agreement (SLA) is established with it. Under this framework, the mobile users may enjoy the cloud services with good QoE regardless of spectrum scarcity. However, these researches mainly takes what, when and how to offload the task from the mobile to cloud. In Flores et al. [18], the main consideration is how to offload the tasks to the cloud in real situation. Provided with stable support from the cellular network, smartphone can offload the computing task to the remote cloud at any time and any places. The advantage of this mode is high reliability in the service supply while the disadvantage is the high cost and delay of the cellular network [19].

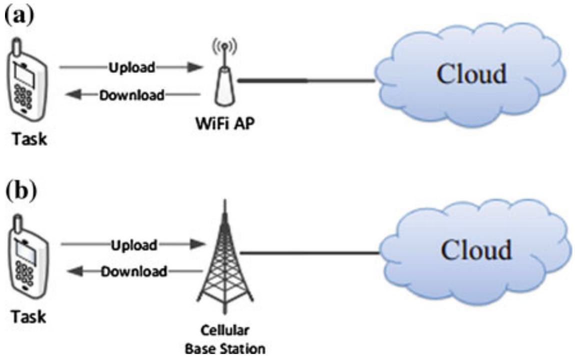


Fig. 6.1 Illustration of task offloading through remote cloud service mode: **a** remote cloud service mode via WiFi; **b** remote cloud service mode via cellular networks

6.2.2 Task Offloading Based on Mobile Cloudlets

The concept of cloudlet was presented in Satyanarayanan et al. [20] and then discussed in Miettinen et al. [21]. These cloudlets are described as "data center in a box". Nowadays, discussions on cloudlets focus on the definition of the cloudlet size, lifetime of cloudlets node life and available time to solve a basic problem of the cloudlets: under what condition is it feasible for the mobile cloudlets to provide mobile application service [6]. Moreover, Wang et al. [22] has proposed a kind of opportunistic cloudlet offloading mechanism based on mobile cloudlets and Truong-Huu et al. [23] has proposed a kind of stochastic workload distribution approach based on mobile cloudlets. However, only when task node is connected with service node, can the task offloading be allowed. After the D2D connection is built up between the task node and service node, the energy cost is economic since the content delivery are carried out through the local wireless network (i.e. WiFi and bluetooth). Zhou et al. [24, 25] firstly propose a distributed information-sharing strategy with low complexity and high efficiency. The limitation of mobile cloudlets lies in the strict requirement on time because the task node and service node shall have enough time to offload the computing tasks and treat the feedback. Once the task node and service node disconnect due to high user mobility and other factors of network dynamics while task offloading is not completed, the computing will fail.

6.3 Opportunistic Task Scheduling Over Co-located Clouds Mode

6.3.1 Motivation

Along with the rapid development of wireless communication and sensor technology, the mobile devices are equipped with more and more sensors, as well as powerful computing and perception abilities. Under such background, the crowdsourcing application emerges as a new type of mobile computing: a large number of users utilize mobile devices as basic sensing units to achieve distributed data, collection and utilization of the perception tasks and data through mobile Internet to complete even larger and more complicated social perception tasks. The participants who complete the complicated perception tasks with crowdsourcing do not need professional skills. The crowdsourcing has succeeded in the applications of positioning, navigation, urban traffic perception, market forecasting, opinion mining, etc. which are labor-intensive and time-consuming. Based on the vast quantity of common users, it distributes tasks in a free and voluntary manner to common users and let them complete the tasks that they can never complete independently. The idea of crowdsourcing also has broad applications for task offloading [18, 26].

In this chapter, the task offloading is realized by remote cloud and mobile cloudlets. We consider that either traditional remote cloud or mobile cloudlets exhibits a certain limitation during task offloading, especially under limited bandwidth. Given the application of image segmentation as a typical scenario (see Sect. 6.4.1 for details), the size of the picture taken by a mobile device is generally large. However, the user only care some specific region of interest (ROI). For example, the interest of some users towards a whole picture is only the face image appearing in the picture. Compared to the size of the picture, the size of such ROI is much smaller. In order to achieving energy saving, it's beneficial to finish the task of image segmentation locally. However, the transmission of the whole picture to the cloud is a must using remote cloud service mode. In comparison, the energy cost for offloading the task to the cloud through the cellular network can be eliminated in either mobile cloudlets service mode or OSCC mode. However, the use of mobile cloudlets service mode incurs the limit on user's mobility. Thus, how to design an optimal solution to minimizing energy cost while guaranteeing high user's QoE is a challenging issue.

6.3.2 OSCC Mode

In mobile cloudlets, user mobility or network dynamics make contacting time of two users short, which decreases the probability of task completion. However, we assume that the contacting time via D2D link is enough for a task node to transmit content associated with computation to a service node. When the task node and the service node disconnect, the computing of the service node will still carry on until the sub-tasks complete. We call the new service mode as OSCC. A basic feature of OSCC is that the contact between the task node and the service node can be either short or long instead of limiting users' mobility to guarantee the contact time for task completion in conventional cloudlet based service mode. We assume each computing task has a deadline, before which the computing result should be returned from the service node to the task node. Based on the location of the service node upon the sub-task completion, there are three situations: (i) move close to the task node again within D2D communication range, (ii) cannot to connect with the task node directly by D2D communications, but WiFi can still work, (iii) in no way to connect the task node by neither D2D links nor WiFi, but the cellular network can still work.

Considering three situations above, the OSCC service mode was classified into the following three categories.

- OSCC (back\forth): Wang et al. [22] have proposed a task offloading method with the help of cloudlet, used the statistical law of the node movement and calculated the probability of the meeting of the task node and service node for twice at least. In that way, before the completion of the required computing tasks, once the service node meets the task node again and the sub-tasks of the service node have finished, the result of the sub-tasks can be transmitted to the task node successfully. We call the task offloading service mode by mobile cloudlets as "back-and-forth service in cloudlet". However, in this mode, user mobility is always limited to ensure the second meeting between the task node and the service node. Even though, the mobility support of OSCC (back&forth) mode is higher than remote cloud mode through WiFi. Therefore, we mark the mobility support level of OSCC (back&forth) as the mobility support should be leveled as "Medium" in Table 6.1.

- OSCC (one way-WiFi): Considering that the service node may move another cell, where WiFi is available, for example, the owner of the service node comes back to his/her home, the sub-task result can be uploaded to the cloud through WiFi. Generally, the data size of sub-task result $\left(S_{sub-tk}^{result}\right)$ is smaller than the original size of the data associated with the sub-task $\left(S_{sub-tk}^{recv}\right)$. Let r denote the rate of S_{sub-tk}^{result} over S_{sub-tk}^{recv}. With the decrease of r, OSCC (one way-WiFi) outperforms remote cloud service mode more.
- OSCC (one way-Cellular Network): In this mode, the economic way for computing task result feedback is not available. That is, the service node moves to a place without WiFi, it needs to upload the sub-task result to the cloud through cellular network. As for the r value, the small the r is, the better the effect of OSCC mode is.

A typical example is presented to explain the above mentioned three kinds of OSCC service modes, as shown in Fig. 6.2. David has a computation-intensive task which cannot be carried out only by his mobile phone. Within his D2D communication range, the phones of David's three friends, Smith, Alex and Bob are all in idle state. So, David divides the computing task into 3 sub-tasks and transmits them

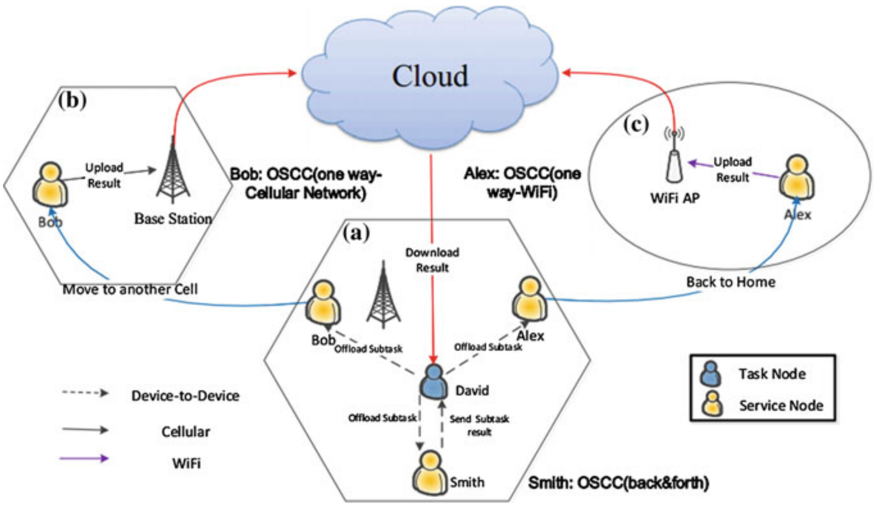


Fig. 6.2 Illustration of the task offloading at OSCC mode: **a** Smith send sub-task result to David via D2D connection; **b** Bob send sub-task result to cloud via 3G; **c** Alex send sub-task result to cloud via WiFi

to the three phones via D2D links. Smith is good friend of David and moves together with David, so he always keeps contact with David. After the completion of the task, the result of his sub-task computation will be transmitted to David directly through D2D connection. We assume Alex goes back to home with available WiFi link, so OSCC (one way-WiFi) service mode is used. Let's assume that Bob has moved to another cell before the completion of the sub-task, so he uploads the sub-task result to the task node through OSCC (one way-Cellular Network) service mode.

OSCC is quite efficient in some applications, for example, the data size associated with computing task is huge but the result data is relatively small. We consider the example of image segmentation mentioned in Sect. 6.3.1. Compared with remote cloud service mode, OSCC mode can transmit the whole picture through D2D which needs less bandwidth and energy. Compared with mobile cloudlets service mode, OSCC mode features a higher expand ability, as it does not require the task node keep contact with service nodes through D2D communications all the time or within a region, so it provides high freedom for the task node and service node. Therefore, OSCC mode which can be taken as the compromised mode between remote cloud and mobile cloudlets, achieving more flexibility and cost-effectiveness. In order to understand how to use this new task offloading mode better, we establish a mathematical model and provide solutions to some optimization problems. As for the OSCC mode, here we give the hypothesis as follows: (i) The computing tasks can be divided into multiple sub-tasks. According to different applications and the properties of the task, we classify the division of the task into two categories, i.e., cloned task and non-cloned task. (ii) Each service node will not accept the same cloned task more than once. (iii) Packet loss is not considered during the transmission of network data.

6.4 OSCC Mode

Assume that there are M mobile nodes in the mobile cloud computing network. Let N denote the total number of task nodes and n denote the amount of sub-tasks for a task node. Task node can communicate with service node only when they are within the transmission radius R. That is, task node cn_i and service node sn_j can communicate if $\left\|L_i(t) - L_j(t)\right\| < R$, L_i and L_j are the positions of the two nodes at time t. The node mobility is i.i.d. Typically, the inter-contact duration of any two nodes follows exponential distribution with parameter λ [27, 28]. Thus, λ reflects the average meeting rate of two nodes. The probability without contact within Δt time can be calculated as $P\{t > \Delta t\} = e^{-\lambda \Delta t}$.

The task node have a total amount of computation task Q which can be divided into n sub-tasks. Q can be denoted as:

$$Q = \sum_{i=1}^{n} x_i \tag{6.1}$$

where x_i is the workload assigned to node i. Let's assume that service node sn_i have a per unit process speed v_i and this service node can process sub-task x_i. Table 6.2

Table 6.2 Variables and notation of OSCC mode

Variable	Default value	Explanation
M	500	Number of nodes in the cell
cn_i	N/A	A task node with index i and have computation task to be executed
sn_k	N/A	A service node with index k, which serves as available resource for computation offloading
N		The total number of task nodes in the cell
n		The amount of sub-tasks for a task node
K		Number of total sub-tasks in the cell
$X(t)$.	N/A	The number of service nodes at time t
$S_i(t)$	N/A	The function of number of sub-tasks assigned for a task node cn_i at time t
λ	0.0001	Average meeting rate of two nodes in the cell
$r_{t,t+\Delta t}(i)$	1/0	Whether sn_i assigns sub-task successfully within Δt
$\theta^i_{t,t+\Delta t}(k)$	1/0	Whether service node sn_k gets assignment of sub-task for cn_i
t^*	N/A	The average time to complete the computation of a whole task
t^*_s	N/A	t* under computation clone mode
Q	200	Size of total computation task
x_i	N/A	Size of sub-ask the serve node sn_i have
r	0.5	The ratio of S^{result}_{sub-tk} and S^{recv}_{sub-tk}
$E^{cell}_{n\to c}$	2	The per unit communication cost from task node to cloud via cellular network
$E^{cell}_{c\to n}$	2	The per unit communication cost from cloud to task node via cellular network
E^{cloud}_{proc}	0.1	The per unit energy cost for computation tasks processed in cloud
E_{D2D}	1	The per unit communication cost from task node to service node
$E^{node}_{proc}(k)$	0.2	The per unit energy cost for service node sn_k to process a sub-task locally
ρ	0.001	The probing cost per time unit
t_d	4000	Deadline for computation task completion time
v_i	N/A	Per unit process speed of service node sn_i
C_{Cloud}	N/A	The total energy cost for computation task executed in remote cloud
$C_{cloudlet}$	N/A	The total energy cost for computation task executed in CCS mode
C_{OSCC}	N/A	The total energy cost for computation task executed in OCS mode
ω	0.5	A weight factor which indicates the emphasis

describes the notations and default values used in this chapter. In the following section, task duration and energy cost of OSCC mode will be analyzed.

6.4.1 Task Duration

The task duration consists of time consumed by two procedures, i.e. (i) the delivery of task contents, and (ii) task result feedback. For the sake of simplicity, we assume task node knows the capabilities of service nodes. Let t^* denote the task duration. Considering the situation where a task is computation-intensive and WiFi is unavailable, the delay of local processing (denoted by Q/v, where v is processing speed of the task node) is typically larger than t^*.

A task mainly consists of three components, i.e., processing code, data and parameter(s). For a non-cloned task, a task node divides the task into a certain number of sub-tasks. Each sub-task includes a specific combination of processing code, data and parameter(s). Typically, the data contents and parameters between two sub-tasks are different while processing codes probably are identical. For a cloned task, it can be copied during task dissemination. It has intrinsic feature of random parameter-oriented computation. To illustrate the difference between a cloned task and a non-cloned task, the characteristics of a cloned task are detailed as follows:

- When a service node receives a cloned task, it can further duplicate the cloned task and disseminate it to other service nodes. However, a service node will not accept the same cloned task more than once.
- A cloned task typically includes processing code without data and pre-assigned parameters. When a service node receives the cloned task, it executes the processing code with a stochastic parameter generated by the local machine (i.e., the service node).
- The computation complexity of executing a processing code with various stochastic parameters for a bunch of times is the major purpose for a task node to allocate cloned tasks to numerous service nodes.
- Though there exists high redundancy among various cloned tasks in terms of processing code, the computation activities are different in those service nodes handling the cloned tasks.

We further give examples about non-cloned task and cloned task in detail.

Given an example as shown in Fig. 6.3a about non-cloned task, David is the task user (corresponding to task node) and has 20 pictures, which have different images and contain unique ROI in each picture. There are Bob, Alex, Smith and Suri, four users who can reach David through D2D communications. As each person has a different smart phone and specific computing power, the computing task shall be divided into four sub-tasks through “dynamic allocation” which means the four assigned sub-tasks are different from each other. For example, Bob and Alex are

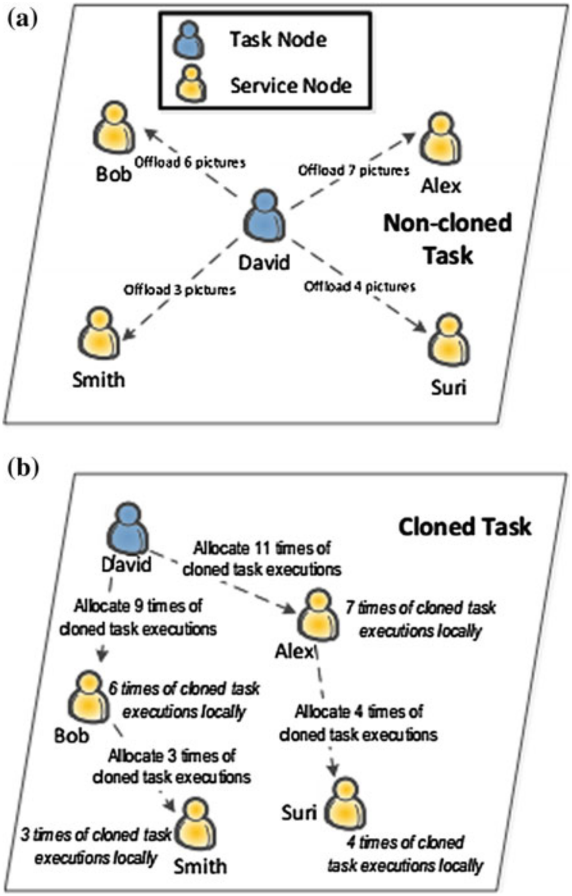

(a)
Task Node
Service Node
Bob
Offload 6 pictures
Offload 7 pictures
Alex
Non-cloned Task
David
Offload 3 pictures
Offload 4 pictures
Smith
Suri
(b)
Cloned Task
David
Allocate 11 times of cloned task executions
Allocate 9 times of cloned task executions
7 times of cloned task executions locally
Alex
6 times of cloned task executions locally
Bob
Allocate 4 times of cloned task executions
Allocate 3 times of cloned task executions
Suri
3 times of cloned task executions locally
Smith
4 times of cloned task executions locally

Fig. 6.3 Illustration of task offloading in opportunistic co-located clouds service: **a** non-cloned task; **b** cloned task

allocated 6 and 7 images respectively while Smith and Suri are assigned 3 pictures and 4 pictures. When the picture is segmented, the ROI (i.e., computation result) will be sent to task node, which is owned by David. In this example, all of the benefits are obtained by David while Bob, Alex, Smith and Suri provide “free services”. Practically, the intrinsic selfish feature of mobile users constitutes the biggest obstacle for task offloading. For example, most users intend to assign tasks to other users while avoiding accepting the sub-tasks allocated to them. This fact may result in failure of OSCC scenario, where most users like to count on others to help them to execute the tasks while reluctant to share computing capacity to others. In order to solve the problem, an incentive mechanism can be designed. For example, Bob contributes computing capacity of his mobile phone to execute David’s sub-task. A certain amount of incentive is sent to him. Likewise, Alex, Suri and Smith get more or less rewards from David according to their workload. Later, their incentives can be used to obtain favors of speeding up their own computing tasks. However, the design of an incentive mechanism to encourage various users for collaborations on task offloading is not the focus of this chapter. We will address this issue in future work.

In the scenario shown in Fig. 6.3b about cloned task, the task can be cloned for required times. For example, David has a cloned task to be processed for 20 times while only Bob and Alex are within his D2D communication scope. Thus, David assigns Bob and Alex to process the cloned task for 9 times and 11 times, respectively. When Bob receives the assignment, he handles the cloned task for 6 times by himself while seeking the help from Smith to process the cloned task for the left 3 times. Likewise, Alex can reach Suri via D2D link, and allocates 4 times of cloned task executions to Suri. In summary, the 20 times of cloned task executions are allocated to Bob, Alex, Smith and Suri for 6, 7, 3 and 4 times, respectively. In this example, when the service node receives a cloned task, the cloned task can be copied and distributed to other service nodes, which is similar with the epidemic model in online social network. Regarding the energy cost caused by the flooding of cloned task, task clone enables less energy cost since more D2D opportunities are available during cloned task distribution than the case of non-cloned task offloading.

6.4.2 Energy Cost

The energy cost is mainly consists of communication cost and processing cost. The communication cost includes two parts, the first one is consumed for offloading task result to cloud (denoted by $E_{n \to c}^{cell}$), the other part is for cloud to feedback computation result to task node (denoted by $E_{c \to n}^{cell}$). E_{D2D} denotes the energy cost via D2D link. The processing cost includes processing energy cost in cloud E_{proc}^{cloud} and in node E_{proc}^{node}. Considering the heterogeneous capability of service nodes in terms computing power, we give two methods to distribute the nodes of sub-tasks:

- As the task node usually has no knowledge about the processing capacity of the service node, we assume that the task node does not differentiate the computing capability of all the service nodes, that is, they have the same processing speed v_i and the same amount of workload $x_i = Q/n$. The task node will distribute the sub-tasks to the service nodes evenly. However, the shortcoming of such assumption is that the service node with largest delay to submit computation result will cause the increase of the task duration.
- Practically, in order to achieve higher delay performance, the task node should not ignore the heterogenous capabilities of service nodes. This motivates us to propose “dynamic allocation” strategy. With the information of the computing capability of various service nodes, we can distribute the sub-tasks in a more intellectual way. For example, if the service node has stronger computing power, it will receive a sub-task with a larger workload.

6.5 Analysis and Optimization for OSCC Mode

Different service modes have both advantages and disadvantages and we hope to achieve flexible tradeoff among various modes to decrease energy cost and delay while meeting the requirements of user's QoE. In the following section, the delay and energy performance will be analyzed, then the optimization framework will be given.

6.5.1 Analysis for Task Duration in OSCC Mode

(1) **Task duration in OSCC mode with non-cloned task**

First, let's analyze the task duration in the case that the tasks cannot be cloned. The task duration consists of time consumptions from two main parts, i.e., sub-task distribution, and computation execution. Sub-task distribution phase is the phase when the task node assigns sub-tasks to service nodes, including transmitting the contents associated with sub-tasks to the service nodes. Typically, computation delay is much smaller than sub-task distribution delay. For the sake of simplicity, only sub-task distribution delay is considered. Let Δt denote a very small time interval, within which there is only one contact at most. As shown in Table 6.2, if $r_{t,t+\Delta t}(i)$ is 1, it means that a task node m_i successfully meets a service node and assigns a sub-task within Δt, vice versa. Thus, $r_{t,t+\Delta t}(i)$ can be defined as follows:

$$r_{t,t+\Delta t}(i) = \begin{cases} 1 & \mathrm{m_i\ assigns\ sub\text{-}task\ successfully\ within\ \Delta t,} \\ 0 & \mathrm{otherwise.} \end{cases} \tag{6.2}$$

Since the inter-contact duration of any two nodes follows exponential distribution, the probability that m_i assigns a sub-task successfully can be expressed as follows:

$$P\{r_{t,t+\Delta t}(i) = 1\} = 1 - \left(e^{-\lambda \Delta t}\right)^{X(t)} \tag{6.3}$$

where $X(t)$ is the number of service node to the time t. Its expectation can be calculated as: $E\left(r_{t,t+\Delta t}(i)\right) = 1 - \left(e^{-\lambda\Delta t}\right)^{X(t)}$, so the number of service nodes which have no sub-task assignments can be computed as:

$$X(t+\Delta t) = X(t) - \sum_{i=1}^{N} r_{t,t+\Delta t}(i) \tag{6.4}$$

We can obtain the expectation about Eq. (6.4):

$$E(X(t+\Delta t) = E(X(t)) - NE(r_{t,t+\Delta t}(i)) \tag{6.5}$$

Letting Δt be close to 0, using the theory of limit, we can obtain the derivation of $E(X(t))$ as follows

$$E'(X(t)) = \lim_{\Delta t \to 0} \frac{E(X(t+\Delta t) - E(X(t))}{\Delta t} = -N\lambda E(X(t)) \tag{6.6}$$

By solving the ordinary differential equation (ODE) (6.6), we can finally get the function $E(X(t))$ as:

$$E(X(t)) = E(X(0))e^{-N\lambda t} \tag{6.7}$$

By solving the inverse function of Eq. (6.8), we can obtain the average time of task duration (denoted by t^*) as follows:

$$t^* = \frac{\ln \frac{M-N}{E(X(t^*))}}{N\lambda} \tag{6.8}$$

Correspondingly, $E(X(t^*)) = M - Nn$. Aforementioned analysis is for the case that all of the computations are considered.

(2) **Task duration in OSCC mode with cloned task**

Now we analysis for the OSCC mode with cloned task. At beginning of our analysis, for the sake of simplicity, let us just consider only one task node. Let $S(t)$ denote the number of service nodes which have sub-tasks at time t. Let $\delta_{t,t+\Delta t}(m_k)$ denote whether m_k gets sub-task assignment within Δt. We can obtain:

$$E(S(t)) = \frac{S(0)Me^{M\lambda t}}{M - S(0) - S(0)e^{M\lambda t}} \tag{6.9}$$

where $S(0) = 1$. Then, the task duration t can be calculated as follows:

$$t = \frac{\ln\left(\frac{S(t)(M-S(0))}{S(0)(M-S(t))}\right)}{M\lambda} \tag{6.10}$$

Furthermore, let's assume that there are N task node, and each sub-task can be cloned.

Let $S_i(t)$ denote the number of service nodes which have sub-task assignments of task node m_i to the time t, then we can calculate $S_i(t+\Delta t)$ as follows:

$$S_i(t+\Delta t) = S_i(t) + \sum_{k=1}^{M-NS_i(t)} \theta^i_{t,t+\Delta t}(k) \tag{6.11}$$

where $\theta^i_{t,t+\Delta t}(k)$ denoted whether service node m_k gets assignment of sub-task for m_i. As for Eq. (6.11), utilizing the methods similar to Eqs. (6.5, 6.6), we can obtain:

$$E'(S_i(t)) = (M - NE(S_i(t)))\lambda E(S_i(t)) \tag{6.12}$$

Then, by solving ODE (6.12), we can compute $E(S_i(t))$ as:

$$E(S_i(t)) = \frac{e^{\lambda Mt}M}{M - N + e^{\lambda Mt}N} \tag{6.13}$$

Finally, by solving the inverse function of Eq. (6.13), we obtain the average time of the task duration for a single task (denoted by t_s^*) as follows:

$$t_s^* = \frac{\ln\left(\frac{E(S_i(t_s^*))(M-N)}{M-NE(S_i(t_s^*))}\right)}{M\lambda} \tag{6.14}$$

Correspondingly, $E(S_i(t_s^*)) = n$.

6.5.2 Analysis for Energy Cost in Remote Cloud Mode, Mobile Cloudlets Mode and OSCC Mode

In this section, we analyze the energy cost performance for various service mode. Let's consider a worst case where WiFi is not available. For simplicity, it is supposed that there is only one task node and its total computing quantity is Q which can be divided into \$n\$ sub-tasks. Since static allocation can be deemed as the extreme case of dynamic allocation, let's focus on the case of dynamic allocation. We divide the whole energy cost chain into three phases, i.e., task associated contents offloading, execution of sub-tasks, and computation result feedback. Then, the total cost of remote cloud based service mode can be calculated as:

$$\begin{aligned} C_{cloud} &= \sum_{i=1}^{n} \left(E^{cell}_{n\to c}x_i + E^{cloud}_{proc}x_i + rE^{cell}_{c\to n}x_i\right), \\ &= Q\left(E^{cell}_{n\to c} + E^{cloud}_{proc} + rE^{cell}_{c\to n}\right) \end{aligned} \tag{6.15}$$

In mobile cloudlets service mode, the major energy cost comes from the use of D2D communications and periodical detection of the surrounding nodes. Then, the total cost at mobile cloudlets mode can be calculated as:

$$\begin{aligned} C_{cloudlet} &= \sum_{i=1}^{n} (E_{D2D}x_i + E_{proc}^{node}(i)x_i + rE_{D2D}x_i) + M\rho t^*, \\ &= Q(1+r)E_{D2D} + \sum_{i=1}^{n} E_{proc}^{node}(i)x_i + M\rho t^*. \end{aligned} \tag{6.16}$$

Let $\vec{X} = \{x_1, x_2, \ldots, x_n\}$ denote the solution of task allocation, thus minimizing the cost can be specified as the following optimization problem:

$$\begin{aligned} \underset{\vec{X}}{\text{minimize}} \quad & \mathrm{C_{cloudlets}} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = Q \\ & x_i \geq 0 \quad i = 1, 2, \ldots, n. \end{aligned} \tag{6.17}$$

The optimization problem is a linear programming problem and can be solved by using a conventional solver, i.e., Matlab.

Considering the mobility of a service node, it may move to some region without WiFi. In this case, the computation result feedback can be classified into two situations: (1) the service node moves back to the proximity of task node and D2D connection is available. Under this situation, D2D link can be used to deliver the result of sub-tasks, which is the case of OSCC (back\&forth). (2) otherwise, the cellular network is the only choice to transmit the result of sub-tasks, which is the case of OSCC (one way-Cellular Network).

We first give the probability P_i, which denotes the chance that service node sn_i meets task node twice. Let $t_{i,1}$ denote the time interval when task node meets service node sn_i for the first time. Let $t_{i,2}$ denote the time interval between the first meeting and the second meeting for task node and service node. Since the time interval follows exponential distribution and i.i.d., Let t_i denote $t_d - x_i/v_i$. According to total probability theorem,

$$P(t_{i,1} + t_{i,2} \leq t_d) = \int_0^{t_i} P(t_{i,1} + t_{i,2} \leq t_d | t_{i,1} = x)\lambda e^{-\lambda x} dx \tag{6.18}$$

where $P(t_{i,1} + t_{i,2} \leq t_d | t_{i,1} = x) = P(t_{i,2} \leq t_d - x) = 1 - e^{-\lambda(t_d - x)}$ Therefore, the $P_i = P(t_{i,1} + t_{i,2} \leq t_d)$ can be calculated as:

$$\begin{aligned} P(t_{i,1} + t_{i,2} \leq t_d) &= \int_0^{t_i} (1 - e^{-\lambda(t_d - x)})\lambda e^{-\lambda x} dx, \\ &= 1 - e^{-\lambda t_i} - \lambda t_i e^{-\lambda t_i} \end{aligned} \tag{6.19}$$

Then, the cost can be calculated as

$$\begin{aligned} C_{OSCC} = \sum_{i=1}^{n} (x_i E_{D2D} + E_{proc}^{node}(i) x_i + r x_i P_i E_{D2D} \\ + r x_i (1 - P_i)(E_{n \to c}^{cell} + E_{c \to n}^{cell})) + M \rho t^*. \end{aligned} \tag{6.20}$$

Thus, the minimum cost can be computed as:

$$\begin{aligned} \underset{\vec{X}}{\text{minimize}} \quad & \mathrm{C_{OSCC}} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = Q \\ & x_i \geq 0 \quad i = 1, 2, \ldots, n. \end{aligned} \tag{6.21}$$

The optimization problem is hard to solve, we divide this problem in two stages. First we maximize *P*, second we minimize the cost in OCS mode.

Generally, the cost for the service node to offload the computing task to the cloud or the cloud feedbacks the result to the task node through cellular network is more than the cost of D2D. Therefore, considering the energy cost and delay in different situations, a compromising method is desired according to the special applications.

- Remote Cloud: If the computing task is highly sensitive to delay and users can afford high cost to reach a higher QoE, using remote cloud (cellular network) is not a bad choice.
- Mobile Cloudlets: If the task node is very sensitive to the communication cost and the service node moves in a small range, then the use of mobile cloudlets is recommended.
- OSCC: If *r* is very small, and the service nodes require maximum node freedom, choosing OSCC is a best solution.

Now, we give the algorithm 1 about how to chose remote cloud, mobile cloudlets and OSCC.

Algorithm C choosing algorithm

```
begin
notation
    $r$ denotes the ratio of $S_{sub-tk}^{result}$ and $S_{sub-tk}^{recv}$;
    $\lambda$ denotes average meeting rate ;
    $C$ is remote cloud or mobile cloudlets or OSCC;
initialization
if situation have stable WiFi then
    use remote cloud through WiFi;
end if
if $r > 1$ and delay sensitive then
    use remote cloud through cellular network;
end if
if $\lambda$ is small and cost sensitive then
    use mobile cloudlets;
end if
if $r < 1$, $\lambda$ is large and maximum node freedom then
    use OSCC;
end if
    Return C;
```

6.5.3 Optimization Framework

Now, we will give the joint optimization for time delay and energy cost. Due to the different impact of time and energy cost, we introduce a weight factor, denoted as ω, which indicates the emphasis on either time or energy cost. Thus minimizing the time and energy cost of a single task node can be specified as the following problem:

$$\begin{aligned} \underset{\vec{X}}{\text{minimize}} \quad & \mathrm{t}^{*} + \omega \cdot \mathrm{C}_{\mathrm{OSCC}} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = Q \\ & x_i \geq 0 \quad i = 1, 2, \ldots, n. \end{aligned} \tag{6.22}$$

We use genetic algorithms to solve above problem. A genetic algorithm is a heuristic algorithm based on the evolutionary theory of genetics and natural selection and can solve this problem. The genetic algorithm is mainly to use the heuristic method to search for the optimal x_i.

6.6 Performance Evaluation

In this section, the proposed OSCC mode will be evaluated. We set the meeting rate λ is 0.00004–0.00032 per second, M is within the range from 300 to 3000, and ρ is set to be 0.001 per second by default based on the previous work [29].

We considers two aspects for the experiment: (1) What kind of impact do task allocation strategies pose on task duration and energy cost? According to the feature of a task, we classify it into non-cloned task and cloned task. The task allocation strategies can be static and dynamic allocation. (2) In order to evaluate our methods, we compare our methods with closely related work. In Chun et al. [13], remote cloud service mode is the major concern. In Li and Wang [6], mobile cloudlets was introduced in detail.

6.6.1 Task Duration

(1) The time consumed by allocating all the sub-tasks with non-cloned task

Because of the relationship among $X(t)$, N, M, λ and n, we need to evaluate the impact of each parameter on the model. In Fig. 6.4a, we fix M to 500; let n be 10, and set λ to 0.0001, while varying N with various values, including 30, 35, 40 and 45. As shown in Fig. 6.4a, task duration increases when N becomes larger.

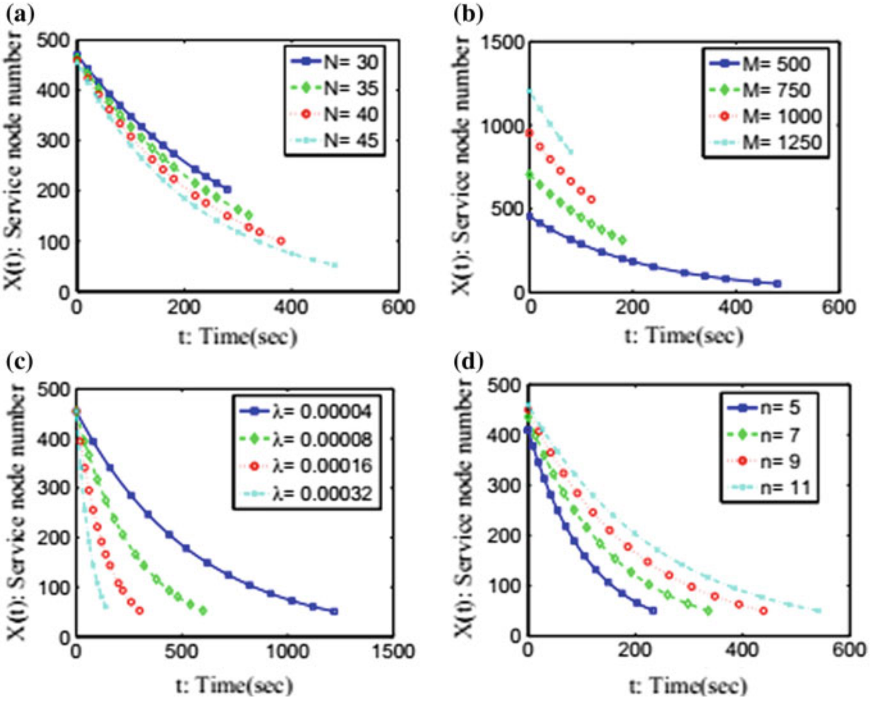


Fig. 6.4 Evaluation on $X(t)$. **a** The impact of N on $X(t)$; **b** The impact of M on $X(t)$; **c** The impact of λ on $X(t)$; **d** The impact of n on $X(t)$

Note that, here, the task duration is the time when all of the users with task achieve their goal of distributing all of the sub-tasks to those mobile users who have no task assignments. From Fig. 6.4a, we also can observe that $X(0)$ is smaller than M. It is because N users already have tasks, thus $X(0)$ is equal to $M - N$.

In Fig. 6.4b, we fix N to 45; n to 10; and λ to 0.00001, while varying M with 500, 750, 1000, and 1250, respectively. As shown in Fig. 6.4b, when $M = 1250$, the task completion time is minimum among all of the scenarios compared. It is because there are more chances for task users to meet a service node to offload task to the node in a shorter period. In comparison, when $M = 500$, $M - N$ service nodes are not enough for consequent task offloading process, and thus causing a larger task duration.

In Fig. 6.4c, we fix M to 500; N to 45; n to 10, while varying λ with 0.00004, 0.00008, 0.00016, 0.00032, respectively, in order to obtain the impact of λ on t and $X(t)$. As shown in Fig. 6.4c, bigger λ represents larger probability for a mobile user to meet with a task node, facilitating the set of sub-tasks to be distributed faster. With the decrease of λ, the task duration increases.

In Fig. 6.4d, we fix M to 500; λ to 0.0001; and K to 450, while varying n with 5, 7, 9, 11, so the N is K/n. As shown in Fig. 6.4d, task duration increases when n

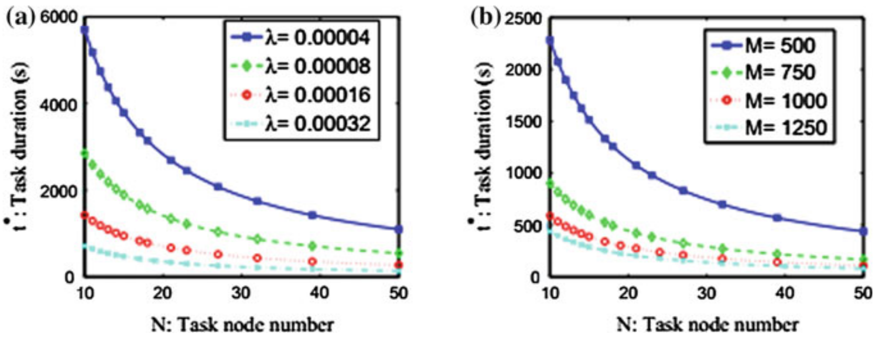


Fig. 6.5 Evaluation on $X(t)$ and t^*. **a** t^*-different λ with varying N; **b** t^*-different M with varying N

becomes larger. It is because when n increases with a fix number of total sub-tasks N is decreased. This indicates that, under the fixed M and λ, the smaller n and the bigger N promote the task completion time. From Fig. 6.4d, we also can observe that $X(0)$ is not equal with each other. It is because when n changes, the N also changes. Similar with Fig. 6.4d, $X(0)$ is equal to $M - N$.

In Fig. 6.5a, we fix M to 500; let K to 450; while varying λ with various values, including 0.00004, 0.00008, 0.00016 and 0.00032. As shown in Fig. 6.5a, as total sub-tasks is fixed, task completion time decreases when N becomes larger. However when N reach 40, this benefit is not distinctive. We also can see that the benefit of increasing N is not significant when the λ is high.

In Fig. 6.5b, we fix λ to 0.0001; let K set to 450; while varying M with various values, including 500, 750, 1000 and 1250. As shown in Fig. 6.5b, like Fig. 6.5a, as total sub-tasks is fixed, task duration decreases when N becomes larger. From Fig. 6.5b, We also can see that the benefit of increasing N is not significant when M reaches 1000.

(2) The time consumed by allocating all the sub-tasks with cloned task

In Fig. 6.6a, we fix M to 500; let n be 10, and set λ to 0.0001, while varying N with various values, including 30, 35, 40 and 45. As shown in Fig. 6.6a, the impact of N on $S_i(t)$ is not distinctive. In Fig. 6.6b, we fix N to 45; n to 10; and λ to 0.0001, while varying M with 500, 750, 1000, and 1250, respectively. As shown in Fig. 6.6b, bigger M represents more chances for a mobile user to meet with a task node. Thus, when M is equal to 1250, $S_i(t)$ increases fastest to reach its maximum of 10. In Fig. 6.6c, we fix M to 500; N to 45; n to 10, while varying λ with 0.00004, 0.00008, 0,00016, 0.00032, respectively. As shown in Fig. 6.6c, bigger λ represents larger probability for a mobile user to meet with a task node. Thus, when λ is equal to 0.00032, $S_i(t)$ increases fastest to reach its maximum of 10.

In Fig. 6.7a, we fix M to 500; let K be equal to 450; we vary λ with different values, including 0.00004, 0.00008, 0.00016 and 0.00032. In Fig. 6.7b, we fix λ to 0.0001; let K be equal to 450; We vary M with different values, including 500, 750,

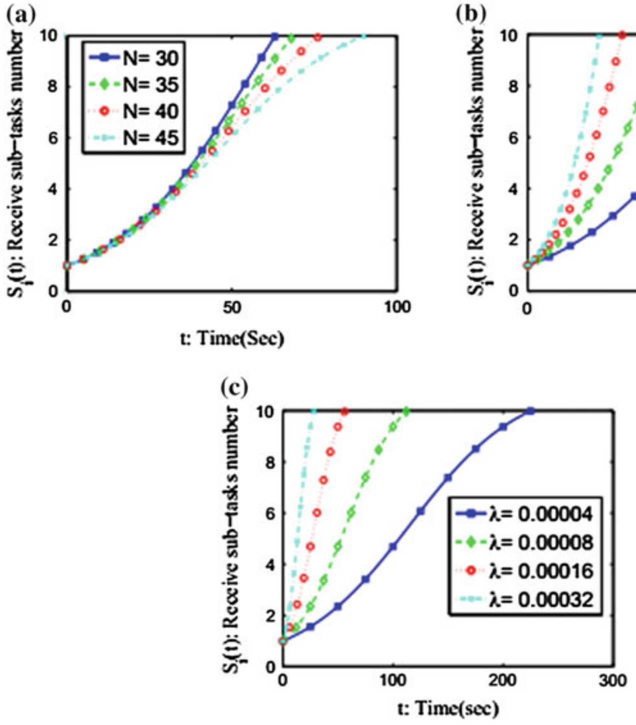


Fig. 6.6 Evaluation on $S_i(t)$. **a** The impact of N on $S_i(t)$; **b** The impact of M on $S_i(t)$; **c** The impact of λ on $S_i(t)$

1000 and 1250. As shown in Fig. 6.7, compared with Fig. 6.5, the task duration t_s^* of Fig. 6.7 is far smaller than the task duration t^* of Fig. 6.5 under the condition of same N and λ or same N and M. It is because task clone is allowed. When task node meet an service node, the service node becomes task node. In other words, the number of task node becomes larger. However, if the task clone is not allowed, the number of task node stay the same.

(3) Task duration in mobile cloudlet mode and OSCC mode

Figure 6.8a has compared the task duration of OSCC mode and mobile cloudlets. From the picture, in the situation that the OSCC can be cloned with a fixed λ, the task duration is the shortest and the task duration of OSCC is shorter than mobile cloudlets. Along with the increase of λ, OSCC presents a better delay performance, because the increase of λ, the task node meets the service node more frequently. As for mobile cloudlets, the task duration decreases gradually from a small λ (for example, from 0.00002 to 0.0001). However, as λ continues to grow, mobile cloudlets task duration starts to increase because of shortened contacting time which

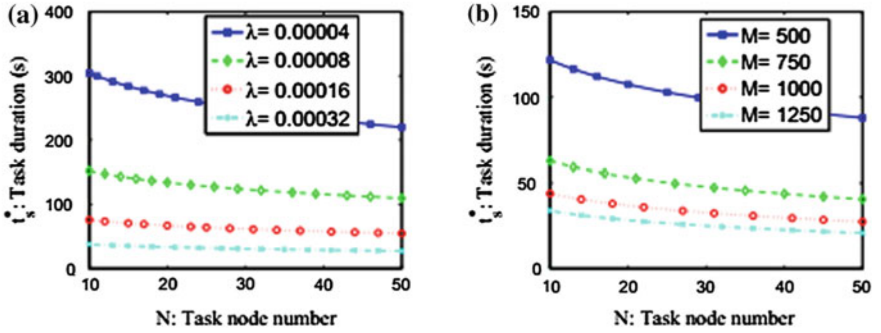


Fig. 6.7 Evaluation on $S_i(t)$ and t_s^*. **a** t_s^*-different λ with varying N; **b** t_s^*-different M with varying N

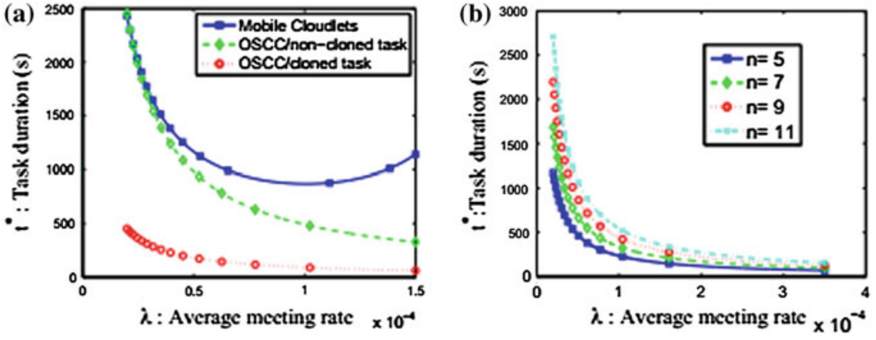


Fig. 6.8 Evaluation on task duration. **a** Compared the task completion time of mobile cloudlets and OSCC mode; **b** OSCC mode task duration-different n with varying λ

leads to inadequate contacting time for the offloading, implementation and feedback of sub-tasks. As shown in Fig. 6.8b, when λ is larger than 0.0003, the performance of OSCC starts to not be distinctive. It is because the contact duration is too short to guarantee a successful sub-task offloading.

6.6.2 Energy Cost in Remote Cloud Mode, Mobile Cloudlets Mode and OSCC Mode

Figure 6.9a has compared the energy cost in the mode of remote cloud and OSCC. Four curves means the energy cost in the mode of remote cloud and the energy costs in the mode of OSCC with different r. As $E_{n\to c}^{cell}, E_{c\to n}^{cell} > E_{D2D}$, when $r < 1$, OSCC is smaller than remote cloud under normal circumstances. However, when $r > 1$, as r increases, The memory consumption in OSCC mode also increases and

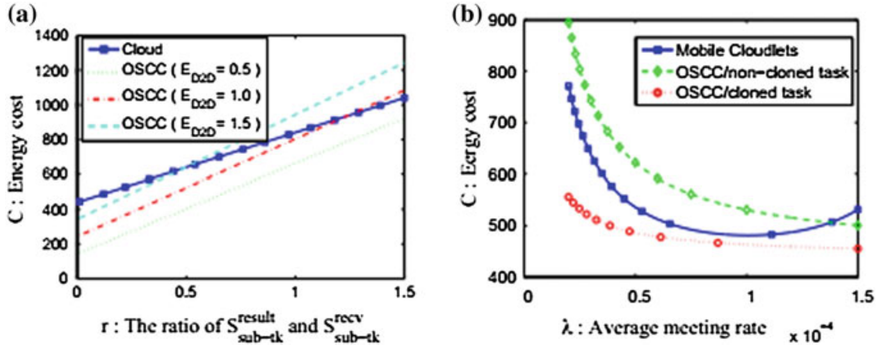


Fig. 6.9 Evaluation on energy cost. **a** Compared the cost between remote cloud and OSCC mode; **b** Compared the cost between mobile cloudlets and OSCC

its increasing speed is faster than remote cloud increasing speed. Moreover, when E_{D2D} increases, the cost of OSCC becomes large.

In Fig. 6.9b, the costs of mobile cloudlets and OSCC are compared with each other. In these three methods, OSCC has appeared smaller energy cost than the other methods under the situation that the computing task can be cloned (i.e., cloned task) when the λ value is fixed, because OSCC can complete the sub-tasks more quickly when it can be cloned. When $0.00002 \leq \lambda \leq 0.00014$, the cost of mobile cloudlets is less than OSCC when it cannot be cloned (i.e., non-cloned task), because OSCC may needs to upload sub-task results to the cloud when the computing task cannot be cloned but mobile cloudlets saves energy accordingly. When λ increases, the contacting time gets shorter, possibly leading to the failure of implementing sub-tasks with mobile device. Therefore, OSCC is better than mobile cloudlets in case of non-cloned task when $\lambda \geq 0.00014$.

6.6.3 *Optimization Framework*

In this subsection, we consider the impact of static and dynamic allocation on the experimental results. The performance of non-cloned task and clone task is also evaluated. Genetic algorithm is used to solve the optimization problem in terms of energy cost and task duration. In our experiments, The weight factor ω about task duration and energy cost is set to be 0.5.

In Fig. 6.10a shows the comparison of cost in term of mobile cloudlets with static allocation and dynamic allocation. As shown in the Fig. 6.10a, the dynamic allocation is almost smaller than static allocation, this is because the task node knows each service node processing cost, so task node send large task to the service node which have lower processing cost. when $\lambda < 0.00005$ and $\lambda > 0.00017$, the benefit of dynamic allocation is not significant.

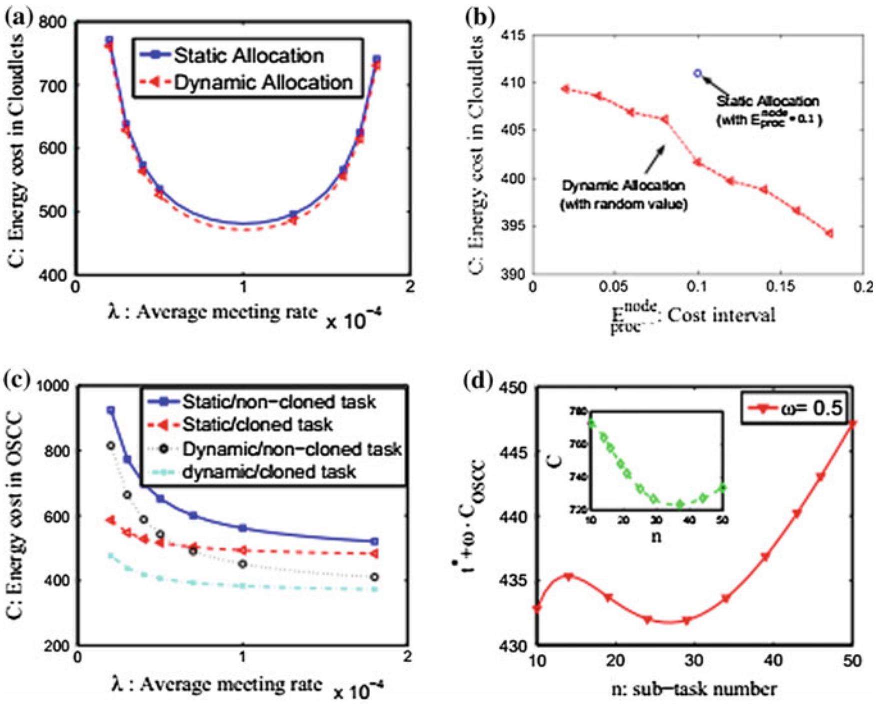


Fig. 6.10 Evaluation on the optimization framework. **a** Compared the cost between static allocation and dynamic allocation in mobile cloudlets; **b** The impact of E_{proc}^{node} on energy cost in OSCC mode; **c** Compared the cost between static allocation and dynamic allocation in OSCC mode; **d** Conjunctive minimization of time and energy cost

In Fig. 6.10b shows the impact of E_{proc}^{node} on energy cost in terms of static allocation and dynamic allocation. In order to verify the effect of dynamic allocation, random value is applied. The circle represents the energy performance of static allocation where E_{proc}^{node} is fixed to 0.1, which represents the same processing capability of service nodes. while the values of data points at X-axis mean the value span where practical value is generated. For example, 0.2 in X-axis means the practical value of E_{proc}^{node} is obtai.ned between 0.01 and 0.19 in a random fashion; 0.01 in X-axis means the practical value varies from 0.09$ to 0.11. As shown in Fig. 6.10b, the larger is the interval, the better performance of dynamic allocation can be obtained.

In Fig. 6.10c shows the comparison of cost in tern of OSCC mode with static allocation and dynamic allocation under non-cloned task and cloned task. We can see that dynamic allocation with cloned task is the smallest energy cost. With the increase of λ, energy cost of all of the compared schemes decreased. In the scheme of dynamic with non-cloned task, the energy cost is decreased with fastest speed. It is because the value of λ have more effect on non-cloned task than cloned task.

When λ reaches 0.00018, the impact of duplicating task becomes smaller. It is because the meeting times increase in unit time slot, and thus speeding up the distribution of sub-tasks.

In Fig. 6.10d shows the conjunctive minimization of task duration and energy cost. we set $\omega = 0.5$. In the embedded figure in Fig. 6.10d, with the increase of sub-task number n and when $n < 35$, the cost decreases. It is because the amount of sub-task allocated to service nodes becomes smaller when total task Q is fixed and more sub-task communication with D2D. However, since the number of sub-tasks is increase, it need more time to deliver the task content and the periodically probing, so the task duration increase. Even more, when $n > 35$, the cost increase since the periodically probing excessive cost due to the task duration. So there exists a trade-off, we try to decrease time and energy cost by obtaining the solution to the optimal function. As shown from Fig. 6.10d, using genetic algorithms, when n is equal to 26, the optimized performance is achieved.

References

1. V. Leung, T. Taleb, M. Chen, T. Magedanz, L.-C. Wang, R. Tafazolli, Unveiling 5G wireless networks: emerging research advances, prospects, and challenges [guest editorial]. IEEE Network **28**(6), 3–5 (2014)
2. N. Fernando, S.W. Loke, W. Rahayu, Mobile cloud computing: a survey. Future Gener. Comput. Syst. **29**(1), 84–106 (2013)
3. L. Lei, Y. Zhang, X. Shen, C. Lin, Z. Zhong, Performance analysis of device-to-device communications with dynamic interference using stochastic petri nets. IEEE Trans. Wireless Commun. **12**(12), 6121–6141 (2013)
4. B. Han, P. Hui, V.A. Kumar, M.V. Marathe, J. Shao, A. Srinivasan, Mobile data offloading through opportunistic communications and social participation. IEEE Trans. Mob. Comput. **11**(5), 821–834 (2012)
5. X. Wang, M. Chen, Z. Han, D.O. Wu, T.T. Kwon, TOSS: Traffic offloading by social network service-based opportunistic sharing in mobile social networks, in *INFOCOM, 2014 Proceedings IEEE* (IEEE, 2014), pp. 2346–2354
6. Y. Li, W. Wang, Can mobile cloudlets support mobile applications? in *INFOCOM, 2014 Proceedings IEEE* (IEEE, 2014), pp. 1060–1068
7. K. Zheng, X. Zhang, Q. Zheng, W. Xiang, L. Hanzo, Qualityof-experience assessment and its application to video services in LTE networks. IEEE Wirel. Commun. **22**(1), 70–78 (2015)
8. D. Candeia, R. Araujo, R. Lopes, F. Brasileiro, Investigating business-driven cloudburst schedulers for e-science bag-of-tasks applications, in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)* (2010), pp. 343–350
9. W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F.A. Silva, C.O. Barros, C. Silveira, Running bag-of-tasks applications on computational grids: the mygrid approach, in *IEEE International Conference on Parallel Processing (ICPP)* (2003), pp. 407–416
10. T. Taleb, A. Ksentini, Follow me cloud: interworking federated clouds and distributed mobile networks. IEEE Network **27**(5), 12–19 (2013)
11. H. Flores, S. Srirama, Mobile code offloading: should it be a local decision or global inference? in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (ACM, 2013), pp. 539–540

12. M. Barbera, S. Kosta, A. Mei, J. Stefa, To offload or not to offload? the bandwidth and energy costs of mobile cloud computing, in *INFOCOM, 2013 Proceedings IEEE* (IEEE, 2013), pp. 1285–1293
13. B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in *Proceedings of the Sixth Conference on Computer systems* (ACM, 2011), pp. 301–314
14. S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in *INFOCOM, 2012 Proceedings IEEE* (IEEE, 2012), pp. 945–953
15. W. Cirne, F. Brasileiro, L. Costa, D. Paranhos, E. Santos-Neto, N. Andrade, C.D. Rose, T. Ferreto, M. Mowbray, R. Scheer et al., Scheduling in bag-of-task grids: the paua case, in *IEEE Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)* (2004), pp. 124–131
16. M. Jia, J. Cao, L. Yang, Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing, in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (IEEE, 2014), pp. 352–357
17. L. Lei, Z. Zhong, K. Zheng, J. Chen, H. Meng, Challenges on wireless heterogeneous networks for mobile cloud computing. IEEE Wirel. Commun. **20**(3), 34–44 (2013)
18. H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile code offloading: from concept to practice and beyond. IEEE Commun. Mag. **53**(3), 80–88 (2015)
19. M. Chen, Y. Hao, Y. Li, C.-F. Lai, D. Wu, On the computation offloading at ad hoc cloudlet: architecture and service modes. IEEE Commun. Mag. **53**(6), 18–24 (2015)
20. M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
21. A.P. Miettinen, J.K. Nurminen, Energy efficiency of mobile clients in cloud computing, in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (USENIX Association, 2010), p. 4
22. C. Wang, Y. Li, D. Jin, Mobility-assisted opportunistic computation offloading. IEEE Commun. Lett. **18**(10), 1779–1782 (2014)
23. T. Truong-Huu, C.-K. Tham, D. Niyato, A stochastic workload distribution approach for an ad hoc mobile cloud, in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)* (IEEE, 2014), pp. 174–181
24. L. Zhou, Specific-versus diverse-computing in media cloud. IEEE Trans. Circuits Syst. Video Technol. **25**(12), 1888–1899 (2015)
25. L. Zhou, Z. Yang, H. Wang, M. Guizani, Impact of execution time on adaptive wireless video scheduling. IEEE J. Sel. Areas Commun. **32**(4), 760–772 (2014)
26. Q. Li, P. Yang, Y. Yan, Y. Tao, Your friends are more powerful than you: efficient task offloading through social contacts, in *2014 IEEE International Conference on Communications (ICC)* (IEEE, 2014), pp. 88–93
27. Y. Li, Y. Jiang, D. Jin, L. Su, L. Zeng, D. Wu, Energy-efficient optimal opportunistic forwarding for delay-tolerant networks. IEEE Trans. Veh. Technol. **59**(9), 4500–4512 (2010)
28. W. Gao, G. Cao, User-centric data dissemination in disruption tolerant networks, in *INFOCOM, 2011 Proceedings IEEE.* (IEEE, 2011), pp. 3119–3127
29. X. Wang, M. Chen, Z. Han, T.T. Kwon, Y. Choi, Content dissemination by pushing and sharing in mobile cellular networks: an analytical study, in *2012 IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS)* (IEEE, 2012), pp. 353–361

Chapter 7
Mobility-Aware Resource Scheduling Cloudlets in Mobile Environment

Abstract The ever-growing number of smart phones is producing explosive amounts of traffic in order to support a wide plethora of multimedia services. A recent Cisco report estimates that global mobile traffic will exceed 24.3 exabytes monthly in 2019.

7.1 Introduction

7.1.1 Mobile Environment of Heterogeneous Network

The ever-growing number of smart phones is producing explosive amounts of traffic in order to support a wide plethora of multimedia services. A recent Cisco report estimates that global mobile traffic will exceed 24.3 exabytes monthly in 2019 [1, 2]. However, due to the centralized nature of mobile network architectures, it is challenging to cope with the rapidly growing mobile traffic along with the limited capacity of the backhaul link. In order to overcome this issue, paradigms called "content-centric networking" (CCN), "named data networking" (NDN) and "content delivery networks" (CDN) [3, 4] have been proposed to handle content-dominated Internet traffic for the radio access networks (front-haul) and the core networks (back-haul).

Furthermore, alongside the use of diverse network resources [5, 6] in terms of communications, caching and computing are becoming the emerging techniques to meet the increasing demand of user QoE (Quality of Experience) in the next generation 5G networks [7–11], especially for the Internet of Things [12, 13] and healthcare systems [14]. In this chapter, we consider a heterogeneous [15] cellular network, which consists of a Macrocell Base Station (MBS), Small cell Base Stations (SBS) (also called small cell BS; also called as pico, pico- or femto-cells as per the size of the cell) and user device. The caching and computing capabilities of SBSs and user terminals will facilitate content sharing and computation offloading.

To illustrate, viral on-line videos are the kind of content that mobile user repeatedly access, which leads us to an assumption that this content could be cached

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_7

and shared at the edge of the network [16–18]. Typically, content caching at the edge of the network can be classified into two categories, i.e., SBS caching (or femto-caching) [19] through femto-cell access points and Device-to-Device (D2D) caching assisted by user terminals [20]. The SBS can be used for content caching, since it is characterized by a high storage capacity and transmission range, and SBS-assisted cache placement has been discussed in previous studies [21]. In addition, by using D2D links, user terminals in the proximity can share cached content without communicating through the MBS in order to reduce communication cost and delay [22]. With the increase of the hardware performance of mobile devices, mobile devices potentially have the storage and computing capacity required for this type of content sharing [23, 24]. Various studies discuss cache placement on mobile devices in the D2D networks [25, 26].

7.1.2 Resource Scheduling in Cloudlet

For applications of mobile device, we could divide applications into pull-based use case and push-based used case. For pull-based use case, it mainly serves the user drive, and taking the unload of user task as the example, we have mainly introduced the computation offloading problems of user under mobile cloudlets in Chaps. 5 and 6. For push-based use case, it mainly serves the network drive, for example, caching popular videos into SBS or mobile device to reduce network load. In this chapter, we have mainly introduced the dispatch of cache contents and the dispatch of computation tasks under cloudlet, which is connected to MBS and SBS, and which is functionally equivalent to cloud broker, as it not only could perceive the mobility, capacity request and storage resource of mobile device, but also could perceive the resources of MBS and SBS, and then dispatch MBS, SBS and mobile device uniformly. For cache, it could maximize the probability of user's acquisition of request contents. For computation, it could reduce the overall energy consumption of system.

The problem of caching placement to maximize the probability that the user can access content in a wireless system where both SBSs [19] and user terminals [27, 28] have caching capability has been studied. However, most existing studies of caching networks ignore user mobility. Instead, it has been commonly assumed that mobile users are always at a fixed location [29].

In this chapter, we investigate the impact of the user mobility on the performance of caching and computation offloading in mobile environment. Then, we propose a joint mobility-aware and SBS density caching placement scheme (MS-caching), taking into account the impact of user mobility and SBS distribution on the caching placement. Moreover, we addressed the SBS and mobile devices' computing power. We discuss the differences and relationships between caching and computation offloading and present a hybrid computation offloading based on MBS computation offloading, SBS computation offloading and D2D computation offloading. Finally, considering the selfishness of mobile users, we discuss an incentive mechanism to encourage content sharing and computation offloading based on network dynamics, differentiated user's QoE, and the heterogeneity of user terminals in terms of caching and computing.

7.2 Resource Scheduling Based on Mobility-Aware Caching

7.2.1 *Caching Model in SBS and User Device*

In this chapter, we present the strategy of caching placement by considering the user mobility and SBS density. We assume that each user will randomly request files from one content library containing l files $\mathcal{F} = \{F_1, F_2, \ldots, F_l\}$, and the files are sorted according to popularity, i.e., ranking from the most popular (F_1) to the least popular (F_m). Let $|F_f|$ denote the size of F_f In addition, it is assumed that the popularity of a content requested by a user follows a Zipf distribution with parameter γ i.e.

$$q_f = \frac{f^{-\gamma}}{\sum_{i=1}^{m} f^{-\gamma}},\ f = 1, 2, \ldots, l. \tag{7.1}$$

where γ stands for the uneven distribution of popularity in these content. As shown in Fig. 7.1, the user can obtain the requested content mainly via four ways listed as follows:

- **Local caching:** When the user requests content, he or she will firstly examine whether or not such content is cached locally. Once such content is confirmed in the local storage, the user will get access to it without any delay.
- **D2D caching:** If the content requested by the user is not cached locally, the user will seek such content among the devices within the range of D2D

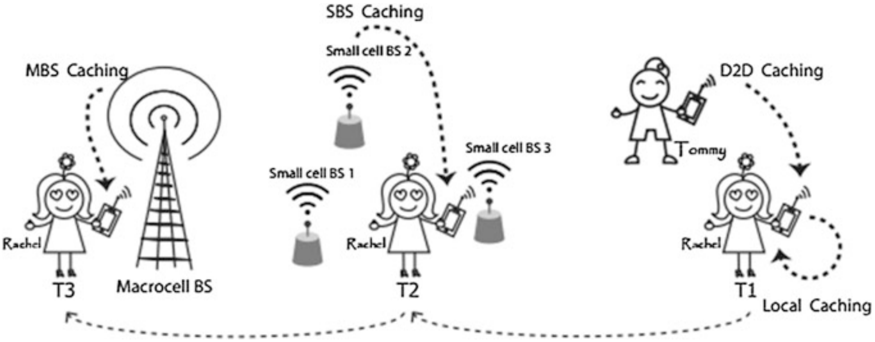


Fig. 7.1 Illustration of the protocol for content access

communications. If there exists one user caching such content, the content will be transmitted to the target user via D2D communications.

- **SBS caching:** Besides D2D caching, if the required content is cached by one SBS, it will be transmitted to the user by the SBS.
- **MBS caching:** If the content requested by the user cannot be accessed in the aforementioned ways, such a request will be forwarded to the MBS, and the content will be delivered to the user by cellular network connection.

7.2.2 Mobility-Aware and SBS Density Caching (MS-Caching)

Given the example in Fig. 7.1, Rachel obtains the requested content by one of the means mentioned above when she moves to different locations starting from time T1 to T3. Due to user mobility, the D2D caching is limited by its short distance range, which presents us with the challenge of how to prepare an optimal cache placement strategy, i.e., content caching at the SBS and the user terminal, and how to maximize the chance to access such content.

Now, let us look at the SBS cache placement. Let R denote the transmission radius of the SBS; CH denotes the cache capacity of each SBS, i.e., the maximum number of files it can store. Following the model in Refs. [30, 31], the SBS spatial distribution is in accordance with Poisson Point Processes (PPPs), and its density is ρ. In terms of cache placement on the SBS, we can describe it as follows: set ωi as the probability of caching a file Fi in the SBSs. Since the SBSs follow PPPs, the probability of at least one SBS caching the content Fi can be calculated as follows:

$$P_i^S = 1 - \mathrm{e}^{-\rho\omega_i\pi R^2} \tag{7.2}$$

Thus, the total probability that a user can get the content from the SBS becomes:

$$P^S = \sum_{i=1}^{l} q_i P_i^S \tag{7.3}$$

If we maximize the probability that the user obtains the content requested under the condition of the storage capacity of SBS, the SBS density-aware caching placement can be obtained as follows:

$$\begin{aligned} \underset{\omega_i}{\text{maximize}} \quad & P^S \\ \text{subject to} \quad & \sum_{i=1}^{l} \omega_i |F_i| \le C_H \\ & 0 \le \omega_i \le 1, i \in \{1, \ldots, l\} \end{aligned} \tag{7.4}$$

For user terminals, we assume that there are N_u mobile devices in this network. Additionally, $\mathcal{D} = \{D_1, D_2, \ldots, D_{N_u}\}$ represents the set of mobile devices. Communication can only be conducted when the shortest distance between any two mobile devices of users is R_{D2D}. Define the inter-contact time $T_{i,j}$ between any two users D_i and D_j as follows:

$$T_{i,j} = \min\{(t - t_0) : \left\|\mathcal{L}_i^t - \mathcal{L}_j^t\right\| < R_{D2D}, t > t_0\} \tag{7.5}$$

where t_0 stands for the moment when the user device D_i just the left communication range R_{D2D} of the user device D_j for the last time. $\mathcal{L}_i^t$ and $\mathcal{L}_j^t$ stand for the locations when the users D_i and D_j are in the moment t. Following the model in Ref. [32], the inter-contact time between any two users D_i and D_j complies with an exponential distribution with a parameter of $\lambda_{i,j}$, which is named as the contact rate of the mobile device i and the mobile device j. Let C_U denote the cache capacity of each user. Let $x_{j,f}$ denote whether the user j caches content F_f. Let T_f denote the deadline to feedback requested content. Thus, within T_f, the probability that the user i obtains the content f via D2D can be calculated as follows:

$$P_{i,f}^{M} = 1 - (1 - x_{i,f})\exp\left(-\sum_{j \in D \setminus \{D_i\}} x_{j,f} T_f \lambda_{i,j}\right) \tag{7.6}$$

Thus, the total probability for the user to get the content through D2D communication becomes:

$$P^M = \frac{1}{N_u} \sum_{i=1}^{N_u} \sum_{i=1}^{l} q_f P_{i,f}^{U} \tag{7.7}$$

If we maximize the probability that the user obtains the content requested under the condition of the storage capacity of mobile devices, the optimal mobility-aware caching placement can be obtained as follows:

$$\begin{aligned} \underset{\mathbf{x_{j,f}}}{\text{maximize}} \quad & P^M \\ \text{subject to} \quad & \sum_{f=1}^{l} x_{j,f} \left|F_f\right| \leq C_U \\ & x_{j,f} \in \{0, 1\} \end{aligned} \tag{7.8}$$

Through joint optimization of PM and PS, the MS caching strategies can be obtained.

7.2.3 Simulation Results and Discussions

We evaluate the probability that users can get contents via simulation results. We compare MS caching with two different caching strategies: popular caching [33] and random caching [20].

- Popular caching: The popular caching strategies on SBSs and on mobile devices of users are as follows: (1) caching strategy on SBSs: most popular content should be stored on each SBS; (2) caching strategy on mobile devices: most popular content should be cached on each mobile device.
- Random caching: The random caching strategies on SBSs and on mobile devices of users are as follows: (1) caching strategy on SBSs: content should be stored at random on each SBS; (2) caching strategy on mobile devices: content should be cached at random on each mobile device.

As for the simulation settings, for simplicity, assume the content size is the same and the value is |F|. The size of content library $l = 30$, and the Zipf distribution parameter $\gamma = 0.8$. The deadline $T_f = 60$ s. The density and transmission range of SBSs are $\rho = 50/\pi 500^2$ and $R = 50$ m, respectively [34]. The system comprises $N_u = 60$ mobile device, and the contact rate $\lambda_{i,j}$ between user D_i and user D_j complies with Gamma distribution $\Gamma(4.43, 1/1088)$ [35]. The caching capacity of SBS and the user terminal is $C_H = 8$ and $C_U = 2$, respectively. For the optimization problem, we utilize the optimization toolkit CPLEX and CVX to solve it. The result is as follows:

- SBS density-aware caching placement: We have provided the relationship between the SBS density and the probability that the user can obtain the requested content. The SBS density-aware caching placement is compared to the popular caching strategy and the random caching strategy, as shown in Fig. 7.2a. When only the SBS is considered, the SBS-assisted cache placement exhibits higher offloading probability than the popular caching and the random caching.
- Mobility-aware caching placement: The user's mobility is closely related to the probability for the user to access the content. The λ is the average contact rate of user devices. Similarly, with the analysis of SBS-assisted caching placement, Fig. 7.2b compares the mobility-aware caching with the popular caching and the random caching. As shown in Fig. 7.2b, the mobility-aware cache placement strategy demonstrates better performance than the random caching placement and the popular caching placement.
- MS caching placement: If we take into account the user mobility and the SBS density, a more advanced cache strategy named MS caching placement can be designed as demonstrated. In Fig. 7.2a, b, we compare the performance of the proposed MS caching placement with other strategies. Since both the SBS density and the user mobility are considered, the MS caching placement obtains the highest probability that users can obtain the contents.

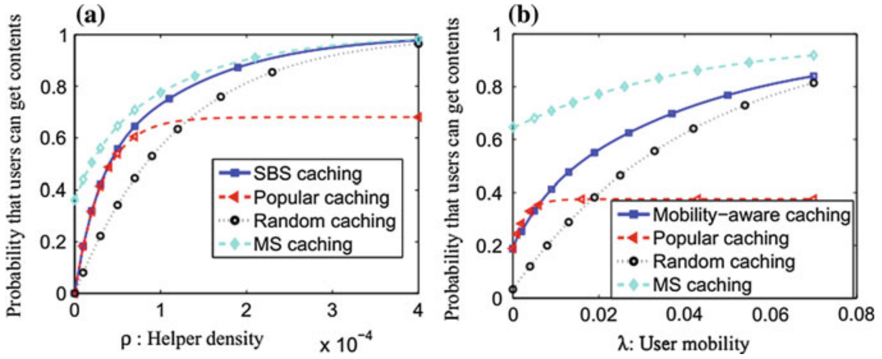


Fig. 7.2 Illustration of the result of caching placement. **a** The impact of ρ on the probability that users can get content; **b** the impact of λ on the probability that users can get content

In Fig. 7.2a, based on the comparison of MS caching, popular caching and random caching, we can obtain the following: (i) as for the density of the SBSs, we cache popular content in a low density region of the SBS, while relatively unpopular content is cached in a high density region to achieve both caching efficiency and a balanced distribution of content; (ii) as for user mobility, the user appears in more locations when his/her mobility is very high, which provides more chances for other users to retrieve the cached content. Thus, a user with high mobility is suggested to cache diverse content, or vice versa, a low mobility user caches popular content.

Based on the above discussions, here, we provide an example of content caching when user's mobility and SBS density are considered. As shown in Fig. 7.3, we differentiate the user mobility in low mobility and high mobility cases. In Fig. 7.3, Rachel sends a request to a file K deadline T_f since D2D-caching and SBS caching are not available. We assume that the MBS knows the users' mobility trajectory in

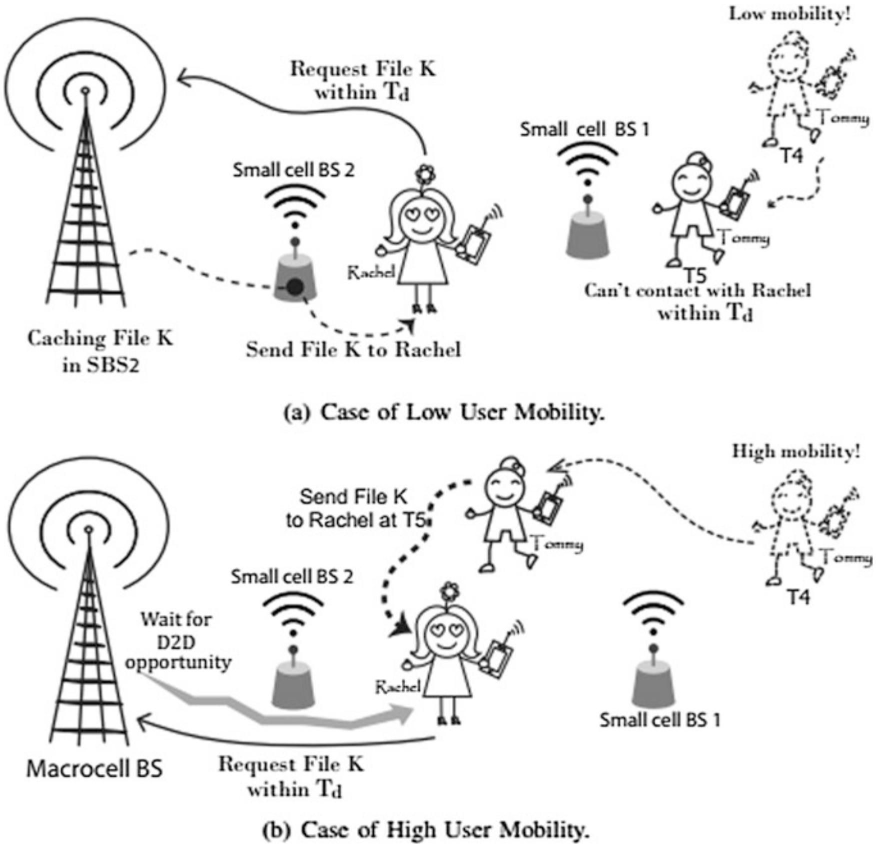


Fig. 7.3 Illustration of the content caching placement. **a** Case of low user mobility; **b** case of high user mobility

the network. If the user mobility is quite low at that moment, the MBS considers the probability for Rachel to meet another user (e.g., Tommy in Fig. 7.3a carrying the file within T_f) is low. Then, The MBS transmits the file K to the SBS closest to Rachel through a back-haul link, and SBS delivers the cached file to Rachel. Figure 7.3b shows the scenario of high mobility, where the MBS predicts that at least one user will likely come into the vicinity of Rachel within T_f according to the mobility status in the network. In response to Rachel's request for the file K, she would wait for the D2D opportunity in order to avoid using a more expensive communication channel (e.g., through femtocell caching). After a short while, Tommy moves to the D2D communication range and sends the file K to Rachel. In the opposite case that Rachel still fails to obtain the requested content while the deadline is soon to expire, the MBS will still utilize the traditional SBS caching.

In future work, we will consider the social relationship of the user terminal in the D2D communication. It can be concluded from the users' social relationship that those with social connections tend to have the same request for content; for example, one region may be divided into different groups, such as an industrial group, a tourism group, a residential group, etc. Different contents will be cached in different regions, and in the same region, the interchange of content may be better achieved.

7.3 Resource Scheduling Based on Mobility-Aware Computation Offloading

7.3.1 Edge Cloud Computing

Mobile cloud computing has been widely studied. Traditional mobile cloud architecture is based on a centralized cloud. For example, in Ref. [36] a cloud-assisted drug recommender system is proposed to provide online medical recommendation based on a centralized cloud. However, with the densification of SBSs to cope with ever-growing data traffic, the weakness of this structure is exposed with higher load and more backhaul delay [37]. As one more consequence, communication cost is also increased to offload computing-intensive tasks to the cloud and return the processed result [38]. To solve the problem, previous work also considered the computing capability of the user terminals and the SBSs [32]. In Ref. [37, 39–41], offloading of the computation task to a mobile-edge cloud is investigated with the consideration of delay and energy cost. By comparison, we address the computation offloading issue by means of using the SBS and the user terminal in the heterogeneous networks while the user's mobility is considered.

7.3.2 Caching Vs. Computation Offloading

We discuss the essential similarities and differences between caching and computation offloading. Content caching is generally provided by the server where the requested content originates from; content is cached during non-peak periods at the MBS, the SBS or the user terminal in order to save the bandwidth in critical time. During "rush hour", corresponding contents are preferred to deliver to the user via SBS or other user terminal. Furthermore caching and computation offloading are correlative; for example, when the user requests for popular videos, the user terminal or SBS will transmit such content to the user, but the content is found not satisfactory in terms of video quality or the format specially required by the user. The user needs to transcode the original format to the one that satisfies the user. Thus, the task will be offloaded to SBS and/or other user terminals to speedup the computation. Table 7.1 provides the main differences between caching and computation offloading.

7.3.3 Hybrid Computation Offloading

We have summarized the methods of mobile-edge computing offloading assisted by MBS, SBS and the user device [37–39]. The edge cloud is called the MBS cloud, when it consists of the computing resources deployed in MBS. Similarly, the edge cloud powered by SBS's computation resources is called the SBS cloud. By comparison, the edge cloud via D2D links is called the mobile cloud.

- MBS computation offloading [39]: A user can offload the computation task to an MBS through a cellular network link. In the research area of mobile cloud computing, when the computation is performed in a cloud environment, the results will be fed back to the user from the cloud via the MBS.
- SBS computation offloading [37]: The computation task is offloaded to an SBS. After SBS completes the computing, the results will be fed back to the user.
- D2D computation offloading [38]: A user terminal can offload the computation task via a D2D link to other mobile devices within the D2D range. Upon the task completion, the results can be transmitted back to the user terminal, if the mobile devices are still within the D2D communication range.

Table 7.1 Caching versus computation offloading

Caching	Computation offloading
No feedback, one-way cache and fetch	Need the feedback of the computation result
The popularity of the cached content is typically high	The popularity of cached computation result can be understood as 0, since it usually only serves one particular user
The size of shared storage is relatively large	The shared space to store the computation result is relatively small

There are some advantages and disadvantages to the above methods. The MBS computation offloading brings the highest communication cost, but provides the largest coverage [38]. The D2D computation offloading has the lowest cost, but it is difficult to ensure the completion of tasks by taking into account the user mobility. The SBS computation offloading falls somewhere in between. Taking into account the advantages and disadvantages of the above three methods, we have proposed a hybrid computation offloading. In the context of the computation offloading, we name the user terminal that has been assigned the computation task as a computation node and the user terminal processing the computation task as a service node. When the computation node and the service node are within range of the D2D communication, the computation node offloads the computational task to the service node. After a period of time, the service node finishes the assigned task; at this moment, the computation node and service node are possibly out of the range of D2D communication because of user mobility. Thus, in some cases, the service node might be required to cache the computing results for a long time until it again comes into the vicinity of the computation node. On the other hand, if a higher storage capacity and a larger transmission radius of the SBS are available, the computing results can be returned back to the computation node in three manners after the computational task is processed at the service node:

- D2D computing result feedback: After the computational task is processed at the service node, the computing results will be returned directly back to the computation node if the service node and the computation node are still within the range of the D2D communication.
- SBS computing result feedback: After the computational task is completed at the service node, the service node will offload the computing results onto the SBS if the computation node is out of the range of the D2D communication. Then, the SBS will transmit the results to the user if it is within the communication range with the computation node.
- MBS computing result feedback: When the result of the computing task has not been transferred to the user before the deadline, namely when the user and the SBS are still not within the communication range, the SBS will upload the results to the MBS, and then, the final results will be passed back to the user.

As shown in Fig. 7.4, Rachel (i.e., the computation node) first divides the computation task into three sub-tasks. Within her D2D communication range, there are three users that can work as service nodes, i.e., Tommy, Eva and Suri. Then, Rachel offloads the three sub-tasks to them via D2D links. When a service node (e.g., Eva) finishes the computation sub-task, it possibly loses D2D connections with Rachel due to the user mobility. Figure 7.4 gives three modes for the computation result feedback, i.e., the D2D computing result feedback, the SBS computing result feedback and the MBS computing result feedback. After the Tommy' sub-task completion, Tommy is still within Rachel's D2D communication range, and the D2D computing result feedback is used. When Eva's sub-task is completed, Eva cannot connect with Rachel via the D2D link; however, an SBS between Eva

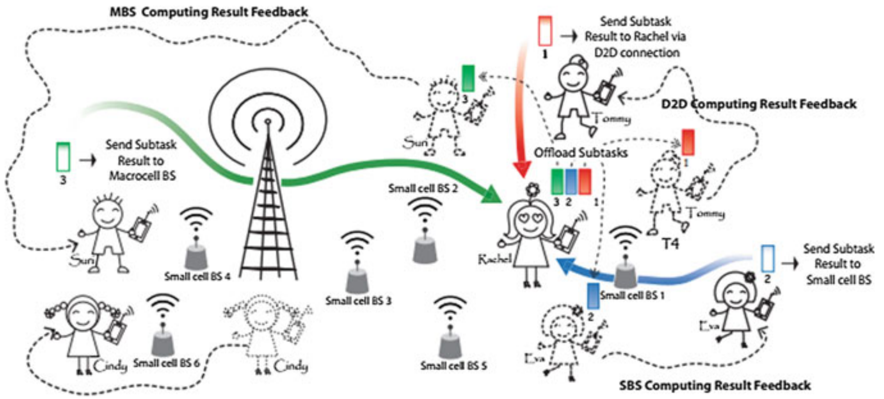

MBS Computing Result Feedback
Send Subtask Result to Rachel via D2D connection
1
D2D Computing Result Feedback
Tommy
Suri
3
Offload Subtasks
3 2 1
Send Subtask Result to Macrocell BS
3
Small cell BS 2
Tommy
T4
Rachel
Send Subtask Result to Small cell BS
2
Suri
Small cell BS 4
Small cell BS 3
Small cell BS 1
2
Eva
Eva
Cindy
Small cell BS 6
Cindy
Small cell BS 5
SBS Computing Result Feedback

Fig. 7.4 Illustration of the hybrid computation offloading: **a** Device-to-Device (D2D) computing result feedback; **b** small cell Base Station (SBS) computing result feedback; **c** macrocell Base Station (MBS) computing result feedback

and Rachel is available. Then, the SBS computing result feedback is used. The worst case is the MBS computation result feedback. Given Suri as an example, he moves far away, and the cellular network link is the only way to feed back the computation result. Based on the above discussion, we can see that the hybrid computation offloading achieves a flexible tradeoff among D2D computation offloading, SBS computation offloading and MBS computation offloading.

7.3.4 Simulation Results and Discussions

We consider four kinds of energy consumptions corresponding to four operations during mobile edge computation, i.e., local computing, mobile offloading, edge cloud computing and downloading of computation results from the edge cloud to mobiles. Here, we mainly consider the energy consumption for the mobile terminal. For the four kinds of computation offloading, they have the same local energy consumption. Additionally, edge cloud computing and downloading of computation results do not consume user terminal's energy. Thus, the major energy consumption of the task is up to mobile offloading. We consider that a user has computation task Q, which can be decomposited into n sub-tasks. That is: $Q = \sum_{i=1}^{n} x_i$. Next, we build up the model to calculate the energy cost of the mobile device. Let $P_t^M(r), P_t^S(r)$ and $P_t^D(r)$ denote the transmission power for the user terminal in terms of the communication via MBS, SBS and D2D, respectively. Let h denote the channel gain and σ_0^2 denote the variable of complex white Gaussian noise. Then, the channel capacity of the user terminal and MBS can be obtained $C^M = B\log\left(1 + \frac{P_t^M(r)h}{\sigma^2}\right)$, where B is the channel bandwidth. Likewise, the channel capacity of the user terminal, SBS and

D2D can be obtained $C^S = B\ \log\left(1+\frac{P_t^S(r)h}{\sigma^2}\right)$, $C^D = B\ \log\left(1+\frac{P_t^D(r)h}{\sigma^2}\right)$. Thus, when obtaining the distance (denoted by r) between the user and MBS, the mobile energy cost for task offloading to the MBS edge cloud is $\mathrm{E}_M = \sum_{i=1}^{n}\left[\frac{x_i}{C^M}\left(\frac{1}{\eta}P_t^M(r)+P_c\right)\right]$, where P_c represents the circuit power consumed at the use terminal. Similarly, with the distance between SBS and the user, the mobile energy cost for task offloading to the SBS edge cloud can be calculated as: $E_S = \sum_{i=1}^{n}\left[\frac{x_i}{C^S}\left(\frac{1}{\eta}P_t^S(r)+P_c\right)\right]$. With higher small cell density, the user has more chance to offload the task onto a small cell with a closer distance and less energy cost. For the case of D2D, if the distance between two adjacent users is known, the D2D energy cost for the task offloading is $E_D = \sum_{i=1}^{n}\left[\frac{x_i}{C^D}\left(\frac{1}{\eta}P_t^D(r)+P_c\right)\right]$. With the increasing of user mobility, the user terminal with a shorter distance will be found for task offloading, which decreases the energy cost. In order to produce optimal performance, the location of task offloading is strategically selected in the hybrid cloud, which exhibits the lowest energy cost. According to [19, 37], we set the total task amount $Q = 10$ Mbytes and $n = 10$. Let $B = 1$ MHz, $\sigma^2 = 10^{-9}$ W, $h = 10^{-5}$. Set the maximum transmit power of the user terminal $P_{max} = 1$ W, and the circuit power $P_c = 115.9$ mW. The result as shown in Fig. 7.5.

In Fig. 7.5, we evaluate the performance of the MBS computation offloading, the SBS computation offloading, the D2D computation offloading and the hybrid computation offloading in terms of communication cost. With the increase of SBS density, the cost of the SBS computation offloading and the hybrid computation offloading decrease since higher SBS density facilitates the computation result offloading to the SBS, as shown in Fig. 7.5a. Figure 7.5b shows the impact of the user mobility on the energy cost. With the increase of the user mobility, both the D2D computation offloading and the hybrid computation offloading exhibit lower

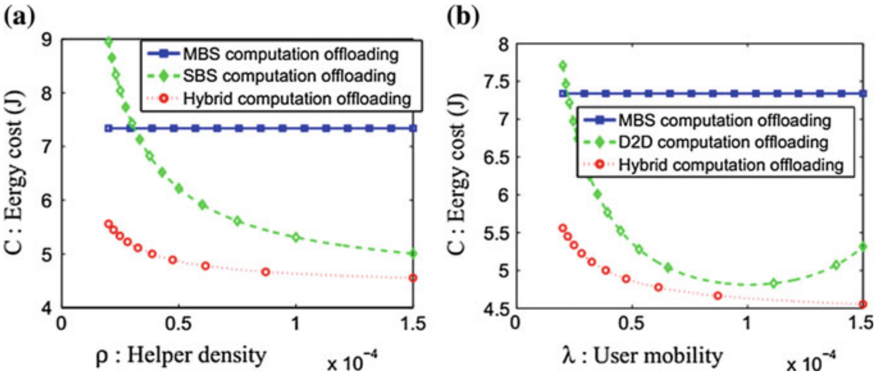


Fig. 7.5 Illustration of the computation offloading energy cost. **a** Comparing the energy cost of MBS, SBS and hybrid computation offloading; **b** comparing the energy cost of MBS, D2D and hybrid computation offloading

energy cost. This is because the probability of the D2D connections increases. The performance of the D2D computation offloading achieves optimization when λ is equal to 0.00011. However, the energy cost increases again when the mobility is too high. This is because the contact time of the D2D connection is too short, which easily causes the failure of the computation result feedback. In comparison, the hybrid computing offloading combines the advantages of the other three computation offloading schemes and produces the lowest energy cost.

7.4 Incentive Design for Caching and Computation Offloading

As already mentioned, the main target of caching and computation offloading in 5G ultra-dense cellular networks is to reduce traffic load and encourages the D2D communications among users. However, the intrinsic selfish feature of user terminals constitutes the biggest obstacle for content caching and computation offloading in practical situations. For example, most users intend to store their favorite files, which at the same time might be also cached by many other users. This fact could result in replicated caching and insufficient use of the accumulative storage space of the network nodes cumulatively. As for computing, most users like to count on others to help them to execute the computation tasks while being reluctant to share computing capacity with others.

In order to solve the problem, this chapter designs an incentive mechanism based on the following three kinds of heterogeneities: (1) the heterogeneity of the user devices, namely each user terminal's storage and computing capabilities are different, which makes some users willing to cache contents and earn incentives through content sharing, while other users prefer to provide computing service to others, and the earned incentive can be used to request cached contents; (2) the heterogeneity of user requirements in terms of user' QoE, namely each user's demand for computing and caching and his/her preference for content are different; (3) the heterogeneity of network conditions, namely the user mobility within the region and the density of the SBSs are different.

Similar to the incentive mechanism for crowdsourcing, more incentives lead to a higher user's QoE. There are two main methods to earn the incentive: cache content and computing tasks for others. Moreover, the two can be transformed into each other; for example, user Bob's mobile phone has large storage capacity to cache more popular content, but its computing capacity is relatively weak; whereas, user Suri's mobile phone has great computing capacity, but weaker storage capacity. When both are within the range of the D2D communication, user Bob may offload some content as requested by user Suri and then get some incentive when the content is sent to Suri. Thereby, user Bob may offload computational tasks to be processed onto user Suri, and user Suri can get some incentive, which can pay for the "debt" for getting the content. Therefore, an incentive balance of content is achieved and replaced by computation. We introduce an incentive mechanism to encourage various users with heterogeneous mobile devices to exchange favors of content sharing and computation offloading.

Specifically, we can divide this incentive into two levels:

- Caching incentive: When the user B obtain content from the user A, the user B needs to pay an incentive (e.g., virtual money), this incentive includes the cost of D2D communication between B and A, the cost of storing content at the expense of the content value from the perspective of the user A. Meanwhile, the user A can get these incentive. From the above, we can see that the popularity of the content of the caching incentive, downloading times of users and caching time are all related to these three aspects.
- Computing incentive: When the user A offloads computing tasks and transfers them to the user B and then the user B helps the user A to proceed with the calculation, the user A will pay the user B an incentive for the communication cost and the computing cost. At the same time, the user B will obtain these incentives. The costs are relatively high due to the fact that the result of computation is equivalent to the content, whose popularity is zero.

References

1. Y. Zhang, M. Chen, S. Mao, L. Hu, V.C. Leung, CAP: Crowd activity prediction based on big data analysis. IEEE Netw. **28**, 52–57 (2014)
2. L. Peng, C.H. Youn, W. Tang, C. Qiao, A Novel approach to optical switching for intra-datacenter networking. J. Lightwave Technol. **30**, 252–266 (2012)
3. G. Fortino, W. Russo, M. Vaccaro, An works. J. Netw. Comput. Appl. **37**, 127–145 (2014)
4. G. Fortino, W. Russo, Using P2P, GRID and agent technologies for the development of content distribution networks. Future Gener. Comp. Syst. **24**, 180–190 (2008)
5. J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, Z. Gu, Online optimization for scheduling preemptable tasks on IaaS cloud systems. J. Parallel Distrib. Comput. **72**, 666–677 (2012)
6. J. Li, Z. Ming, M. Qiu, G. Quan, X. Qin, T. Chen, Resource allocation robustness in multi-core embedded systems with inaccurate information. J. Syst. Archit. **57**, 840–849 (2011)
7. X. Ge, S. Tu, G. Mao, C.X. Wang, T. Han, 5G ultra-dense cellular networks. IEEE Wirel. Commun. **23**, 72–79 (2016)
8. M. Volk, J. Sterle, U. Sedlar, A. Kos, An approach to modeling and control of QoE in next generation networks. IEEE Commun. Mag. **48**, 126–135 (2010)
9. K. Lin, W. Wang, X. Wang, W. Ji, J. Wan, QoE-Driven spectrum assignment for 5G Wireless networks using SDR. IEEE Wirel. Commun. **22**, 48–55 (2015)
10. M.S. Hossain, G. Muhammad, M.F. Alhamid, B. Song, K. Almutib, Audio-visual emotion recognition using big data towards 5G. Mob. Netw. Appl. 1–11 (2016). doi:10.1007/s11036-016-0685-9
11. K. Zheng, X. Zhang, Q. Zheng, W. Xiang, L. Hanzo, Quality-of-experience assessment and its application to video services in LTE networks. IEEE Wirel. Commun. **1**, 70–78 (2015)
12. J. Sterle, U. Sedlar, M. Rugelj, A. Kos, M. Volk, Application-driven OAM framework for heterogeneous IoT environments. Int. J. Distrib. Sens. Netw. 2016 (2016). doi:10.1155/2016/5649291

13. U. Sedlar,M. Rugelj, M. Volk, J. Sterle, Deploying and managing a network of autonomous internet measurement probes: Lessons learned. Int. J. Distrib. Sens. Netw. 2015 (2015). doi:10.1155/2015/852349
14. Y. Zhang, M. Qiu, C. Tsai, M. M. Hassan, A. Alamri, Health-CPS: healthcare cyber-physical system assisted by cloud and big data. IEEE Syst. J. 1–8 (2015). doi:10.1109/JSYST.2015.2460747
15. M. Qiu, E.H.-M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. ACM Trans. Des. Autom. Electron. Syst. 14, 2009. doi:10.1145/1497561.1497568
16. X. Wang, M. Chen, T. Taleb, A. Ksentini, V.C.M. Leung, Cache in the air: exploiting content caching and delivery techniques for 5G systems. IEEE Commun. Mag. **52**, 131–139 (2014)
17. K. Zheng, L. Hou, H. Meng, Q. Zheng, N. Lu, L. Lei, Soft-defined heterogeneous vehicular network: architecture and challenges. IEEE Netw. (2015). arXiv:1510.06579
18. K. Lin, T. Xu, J. Song, Y. Qian, Y. Sun, Node scheduling for all-directional intrusion detection in SDR-based 3D WSNs. IEEE Sens. J. (2016). doi:10.1109/JSEN.2016.2558043
19. K. Shanmugam, N. Golrezaei, A. Dimakis, A. Molisch, G. Caire, Femtocaching: wireless content delivery through distributed caching helpers. IEEE Trans. Inf. Theory **59**, 8402–8413 (2013)
20. N. Golrezaei, P. Mansourifard, A.F. Molisch, A.G. Dimakis, Base-station assisted device-to-device communications for high-throughput wireless video networks. IEEE Trans. Wirel. Commun. **13**, 3665–3676 (2014)
21. J. Song, H. Song, W. Choi, Optimal caching placement of caching system with helpers, in *Proceedings of of the 2015 IEEE International Conference on Communications (ICC),* London, 8–12 June 2015
22. L. Lei, Y. Kuang, N. Cheng, X. Shen, Z. Zhong, C. Lin, Delay-optimal dynamic mode selection and resource allocation in device-to-device communications. IEEE Trans. Veh. Technol. **65**, 3474–3490 (2015)
23. K. Zheng, H. Meng, P. Chatzimisios, L. Lei, X. Shen, An SMDP-based resource allocation in vehicular cloud computing systems. IEEE Trans. Ind. Electron. **12**, 7920–7928 (2015)
24. K. Lin, M. Chen, J. Deng, M. Hassan, G. Fortino, Enhanced fingerprinting and trajectory prediction for iot localization in smart buildings. IEEE Trans. Autom. Sci. Eng. 1–14 (2016). doi:10.1109/TASE.2016.2543242
25. M. Ji, G. Caire, A.F. Molisch, Wireless device-to-device caching networks: basic priciples and system performance. IEEE J. Sel. Areas Commun. **34**, 176–189 (2016)
26. K. Lin, J. Song, J. Luo, W. Ji, M. Hossain, A. Ghoneim, GVT: green video transmission in the mobile cloud networks. IEEE Trans. Circuits Syst. Video Technol. (2016). doi:10.1109/TCSVT.2016.2539618
27. D. Malak, M. Al-Shalash, Optimal caching for device-to-device content distribution in 5G networks, in *Proceedings of the 2014 IEEE Globecom Workshops (GC Wkshps)*, Austin, 8–12 Dec 2014, pp. 863–868
28. M. Ji, G. Caire, A. Molisch, The throughput-outage tradeoff of wireless one-hop caching networks. IEEE Trans. Inf. Theory **61**, 6833–6859 (2015)
29. X. Ge, J. Ye, Y. Yang, Q. Li, User mobility evaluation for 5g small cell networks based on individual mobility model. IEEE J. Sel. Areas Commun. **34**, 528–541 (2016)
30. S.H. Chae, J.Y. Ryu, T.Q.S. Quek, W. Choi, Cooperative transmission via caching helpers, in *Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM),* San Diego, 6–10 Dec 2015
31. X. Ge, B. Yang, J. Ye, G. Mao, C.-X. Wang, T. Han, Spatial spectrum and energy efficiency of random cellular networks. IEEE Trans. Commun. **63**, 1019–1030 (2015)
32. Y. Li, W. Wang, Can mobile cloudletss support mobile applications? in *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications (INFOCOM'14)*, Toronto, 27 Apr–2 May 2014, pp. 1060–1068
33. H. Ahlehagh, S. Dey, Video-aware scheduling and caching in the radio access network. IEEE/ACM Trans. Netw. **22**, 1444–1462 (2014)

34. X. Ge, S. Tu, T. Han, Q. Li, G. Mao, Energy efficiency of small cell backhaul networks based on gauss-markov mobile models. IET Netw. **4**, 158–167 (2015)
35. A. Passarella, M. Conti, Analysis of individual pair and aggregate inter contact times in heterogeneous opportunistic networks. IEEE Trans. Mob. Comput. **12**, 2483–2495 (2013)
36. Y. Zhang, D. Zhang, M.M. Hassan, A. Alamri, L. Peng, CADRE: cloud-assisted drug recommendation service for online pharmacies. ACM/Springer Mob. Netw. Appl. **20**, 348–355 (2015)
37. X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE Trans. Netw. (2015). doi:10.1109/TNET.2015.2487344
38. M. Chen, Y. Hao, Y. Li, C. Lai, D. Wu, On the computation offloading at ad hoc cloudlet: architecture and service models. IEEE Commun. **53**, 18–24 (2015)
39. L. Tong, Y. Li, W.A. Gao, Hierarchical edge cloud architecture for mobile computing, in *Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, 10–15 Apr 2016
40. Q. Liu, Y. Ma, M. Alhussein, Y. Zhang, L. Peng, Green data center with iot sensing and cloud-assisted smart temperature controlling system. Comput. Netw. **101**, 104–112 (2016)
41. X. Ge, X. Huang, Y. Wang, M. Chen, Q. Li, T. Han, C.-X. Wang, Energy efficiency optimization for mimo-ofdm mobile multimedia communication systems with QoS constraints. IEEE Trans. Veh. Technol. **63**, 2127–2138 (2014)

Chapter 8
Machine-Learning Based Approaches for Cloud Brokering

8.1 Introduction

Machine learning is a field of computer science specifically aimed at a challenging goal, quite clearly illustrated by Samuel in 1959, stating that machine learning is that discipline that "gives computers the ability to learn without being explicitly programmed" [1]. Along the years this concept has been sensibly evolved, from the research on pattern recognition to the highly ambitious goal of providing computers with an artificial intelligence.

Nowadays, according to its most diffused interpretation, Machine learning is considered the scientific discipline aimed at dealing with the conception and definition of algorithms and approaches that can learn from, as well as do predictions, leveraging data.

By means of machine-learning based approaches, algorithms are no longer deemed as mere lists of statically defined program instructions derived from a pre-defined, developer-defined, data model. Instead, algorithms become computational entities, able to make data-driven predictions or decisions, through building a model from sample inputs.

Across the years, machine learning has been employed in a wide-range of computer-aided activities, especially the ones in which the explicit design and development of data-agnostic algorithms is unfeasible; classical kinds of applications that have been empowered by machine-learning techniques include spam filtering, optical character recognition, diagnostics [2] and computer vision.

In more recent time, the set of applications making use of machine-learning based approaches significantly increased. This is mainly due to the rise of new technologic paradigms, ranging from physical ones (e.g., mobile devices, IoT, clouds and cloudlets) to software ones (e.g., social networks and media, search engines, media sharing platforms) leading to an unprecedented production of user-generated data, that makes difficult the definition of a pre-defined data model able to make predictions.

C.-H. Youn et al., *Cloud Broker and Cloudlet for Workflow Scheduling*,
KAIST Research Series, DOI 10.1007/978-981-10-5071-8_8

Even more recent approaches are working to the embodiment of machine-learning solutions for supporting the selection and brokering of resources in large computing installment, such as computational clouds and cloud federation.

8.2 Different Ways to Achieve Machine Learning

As a scientific discipline, machine learning is strictly related to computational statistics. Both are focused on conducting deep computing analysis to achieve high-quality predictions. Machine learning has also strong links to mathematical optimization, which actually delivers methodological approaches and theories the field. Machine learning is sometimes combined with data mining to achieve more advanced results [3].

In a nutshell, machine learning can be described, within the field of data analytics, as an advanced method leveraged to develop complex models and algorithms that lend themselves to prediction.

The analytical models defined under the umbrella of machine learning allow data scientists, engineers, and analysts to achieve "reliable, repeatable decisions and results" and unfold "hidden insights" by means of learning from relationships and trends present in the data.

As reported in Table 8.1—Different flavors of machine learning, machine learning approaches are used to be classified into three main categories [4], differing by the nature of the approach available to a learning system to actually learn what to do, depending the available information and data.

Table 8.1 Different flavors of machine learning

Supervised learning	Unsupervised learning	Reinforcement learning
With Supervised learning the machine is fed with sample inputs and the outputs wanted, given by a "teacher" (or supervisor). The objective of the approach is to learn a general rule able to effectively associate inputs to outputs	Unsupervised learning assumes that no labels are provided to the learning system, that is fully in charge to find a certain structure within the input provided. The exploitation of unsupervised learning to achieve a machine-learning based solution is twofold. It can be either the ultimate objective of the process (recognition of hidden patterns) or a methodological tool to achieve a different end (feature learning)	Reinforcement learning is a further approach to machine learning assuming that a computer program would interact with a target environment in which it is expected to achieve a certain, pre-defined goal, without a teacher explicitly telling it whether it has come close to its goal. A pretty typical example of reinforcement learning is to learn to play a game by playing against an opponent. By means of the "expertise" achieved during the different matches, the system learn how to win the game

8.3 Different Methodologies for Machine Learning

Besides three main different "flavours" in which machine learning can be provided, as reported in the previous section, there exist many different technologies, methodologies and strategies to actually achieve, from an operative viewpoint, machine learning solutions. In the following of this section are briefly reported the most diffused approaches.

Association rule learning
This way to conduct machine-learning, consists of a method for discovering relations between large sets of entities [5, 6]. This approach is exploited to identify association rules existing in data by using some measures of attractiveness. Traditionally, such information has been used as the basis for decisions about marketing and placement activities in large grocery stores or supermarkets. Nowadays, association rules are employed for a broader range of tasks in different application areas, such as web-mining.

Artificial neural networks
An artificial neural network [7] machine learning solution is a learning algorithm which main structure is, somehow, inspired by the structure of "real" biological neural networks. Neural-networks based computations are structured as a set of interconnected artificial neurons, aimed at processing information by means of a connectionist approach to computation.

Neural networks are nowadays usually used to model particularly complex relationships between input and output data, to understand the statistical structure characterising an unknown joint probability distribution between variables.

Bayesian networks
A Bayesian network [8] is a probabilistic model symbolised by means of a direct acyclic graph (DAG), representing a collection of random variables and their conditional. In fact, from a formal perspective, Bayesian networks are DAGs in which the nodes represent random variables. Each edge, instead, represents a conditional dependency: not connected nodes represent conditionally independent variables. The typical example used to illustrate a Bayesian network is the probabilistic relationships between users and applications. For a set of given applications, the Bayesian network can be used to compute the probabilities of the presence of actual users.

Clustering
Cluster analysis [9] consists in the association of a set of items into *clusters.* All the items within the same cluster are close, one each other, according to a predefined measure, whereas if items differ, will be located into different clusters. There exist many different techniques for clustering, each focused on different assumptions on the structure and nature of the data. Such characterisation is usually given by means

of some closeness measure and evaluated taking into account the *internal compactness* and *separation* between different clusters. Clustering is one of the most diffused methods for conducting unsupervised learning.

Decision tree learning

This approach [10] implements a predictive model by means of a decision tree mapping the **observations** related to a specific item to **conclusions** about the item final value, respectively represented by branches and leaves. These models represent either classification or regression trees. The former type includes the ones in which the target variable can take a limited set of values, the latter ones represent trees where the target variable can take continuous.

Deep Learning

Deep-learning [11–13] based approaches focus on attempting to model high-level abstractions existing in data by means of a "deep" graph structured on several different processing layers. Deep learning belongs to a set of learning solutions based on feature learning over data. A very key potential of deep learning is to replace human/manually expressed features with efficient algorithms for unsupervised or semi-supervised feature learning. In fact, research in this area is aimed at providing better representations and create models to learn such representations starting from large-scale uncharacterized data. As it happens with artificial neural networks, some of the representations used in the deep learning field, have been inspired by advances in neuro-sciences and are somehow based on patterns for the information interpretation and communication as happens in a nervous system.

Inductive logic programming

Inductive logic programming [14] is an approach to "rule learning" that makes use of logic programming as a way to give uniform representation for input samples. Once an encoding able to represent both the background knowledge and a set of facts is given, an Inductive Logic Programming system is able to derive a hypothesized program that satisfies all positive examples and, at the same time, no negative examples.

This approach is exploited also with non-logic programming paradigm, and in that case, is simply referred as Inductive programming; the most diffused, non-logic, approaches to such field.

Representation learning

In unsupervised learning, it raises quite often the necessity of dealing with notable amount of sparse and complex data. In this perspective, one of the key activities to be conducted is on giving good representations to the input data provided during training. Representation learning [15] algorithms are aimed at to preserve the input information but transforming it to ease their exploitation, usually as a pre-processing step before performing the actual analysis, allowing to rearrange input data that follows a distribution that is not known a priori.

Similarity and metric learning

Similarity and metric learning [16] is an approach to machine learning aimed at determining if elements are similar "enough". According to the classical problem definition, the learning machine is provided with pairs of sample data that are considered similar, and pairs of less similar objects. The machine then needs to learn a similarity function able to predict if new objects received in input are similar. One of the most diffused applications is on recommender systems, to identify items to suggest depending on the past "browsing" history of users. In this set of solutions falls the learning-to-rank approach for the definition of ranking models for information retrieval systems.

Support vector machines

Support vector machines [17] are a set of related supervised learning methods used for data classification and regression. Given a set of input training examples, each one marked as strictly belonging to one of two categories, a support-vector-machine based training algorithm works by building a model able to predict whether a new input example belongs to one category or the other.

Genetic algorithms

A genetic algorithm [18] is a heuristic for searching suitable solutions. This approach works by simulating the process of natural selection, leveraging mechanisms such as "crossover" and "mutation" to create new genotypes for finding high-quality solutions to an input problem. Genetic algorithms are exploited in machine learning, since the nineties. However, in recent times, a cross-fertilization activity took place between the two research fields that leaded machine-learning based techniques to be used to improve the performance of genetic and evolutionary algorithms.

Rule-based machine learning

Rule-based machine learning [19] refers to a broad sector that includes several machine learning methods that achieve an "intelligent" behaviour by identifying, learning, or updating "rules" driving activities aimed at storing, manipulating or applying, knowledge. The key aspect of a rule-based machine-learning based system consists in the identification and exploitation of a collection of relational rules aimed at collectively representing the knowledge base built. Some of the most known rule-based machine learning approaches include learning classifier systems and association rule learning.

Learning Classifier Systems

In the broader area of rule-based machine learning systems, is included the family of learning classifier systems [20], namely, algorithms that put together a detection component and a learning component. Such modules are aimed at identifying a collection of context-dependent rules that collectively store and apply knowledge in a piecewise manner (i.e. by defining specific rules, each devoted to a certain specific range of data) in order to make predictions.

8.4 Machine Learning and Cloud Brokering

As we outlined in the introduction of this chapter, in recent years, machine learning is getting momentum, and its exploitation is becoming of paramount importance also in fields in which it was not exploited before. In the context of application placement and resource brokering, ML-based classification is going to be exploited to assign virtual machines to data centres by ranking each datacentre in accordance with its ability to satisfy a given QoS.

The ranking problem in ML is usually referred to as learning to rank (L2R), falling in the family of the similarity metrics learning systems. Learning to rank solutions provide a ranking of data items according to defined objectives. A learning-to-rank based function works by ranking a set of candidate objects according to their relevance to a given query.

The advantages deriving from the exploitation of machine-learning based solutions is the ability of learning "how-to-rank" from a ground-truth composed of many training examples, instead of relying on user- or developer-provided models.

In fact, once learned, the scoring function provided by the learning-to-rank algorithm is able to approximate the ideal ranking from the examples belonging to the training set. This is a particularly interesting feature/building block to realise brokering solutions, especially in dynamic and complex environment. In fact, by means of a machine-learning approach it is possible to achieve.

8.5 The Current Landscape of Machine-Learning Enabled Cloud Brokering Approaches

In the area of application and resource brokering, across the years, there have been many proposals for making brokering systems "more intelligent" or "smarter". From this perspective, intelligence means the ability of adapting the peculiar choices usually conducted by brokers to find the best resources, when the space of solutions changes depending on mutating conditions affecting both resources and applications.

Beforehand, we presented the most used approaches for conducting machine learning. Only a few of them have been proposed so far in the specific area of application and resource brokering, and even fewer in the area of Cloud Brokering. In the remaining of this chapter are reported the most relevant solutions existing so far in the scientific literature. In order to exemplify the exposition, we focused more on clearness than on completeness, i.e., goal is to give to the reader a clear perspective on the existing solutions instead of aiming at providing a fully comprehensive report of all the existing solutions.

8.5.1 Machine-Learning Based Application Placement in Cloud Federation

Cloud infrastructures and technologies are getting their momentum and their diffusion and exploitation is widening. Nowadays, they are used in many different sectors and areas, following different deployments and needs. Such widely differentiated set of shapes in which clouds are provided to final users and companies led to new challenges and the definition of new requirements. In turn, such needs generated new kind of infrastructures and technologies. Among them is of particular interest the idea of cloud federation [21–23].

Cloud federation are characterised by the ability of federating multiple cloud installations to be able to provide wide area coverage of services, where a single installation is not enough. Federation can be either composed by clouds owned by different providers or by a single one, as happens with the different availability zones of Amazon.

From this perspective, the brokering activity performs a pretty different goal, in fact it is no longer focusing on the actual allocation of VMs to physical machines, instead it is aimed at detecting the best cloud installation for a given VM, depending on a wide set of features, including the location of the requests. A notable result in this field has been proposed by Unuvar et al. [24], in their work the authors focus on the selection of the best availability zone for hosting a VM by exploiting machine learning techniques to maximise user satisfaction.

From a bird-eye-view perspective, the system they propose has a very straightforward structure; its architecture is depicted in Fig. 8.1. As can be observed, it holds the assumption that in the availability zones there are monitoring tools deployed within it aimed at providing the necessary measurements for evaluating the satisfaction level of each requirement specified in the request. Such monitoring support

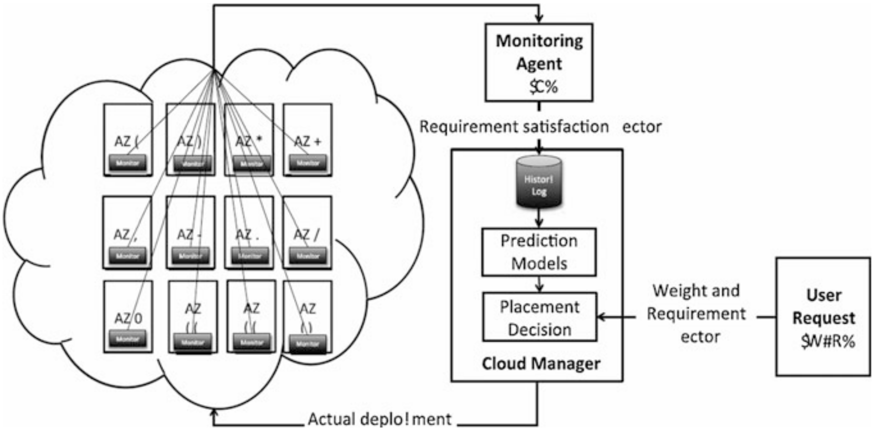


Fig. 8.1 System diagram of the approach proposed by Unuvar et al. [24]

gathers a wide range of information about the architectural features of a node, such as failure and recovery notifications, runtime performances such as throughput of various resources, etc. To this end it can be either exploited an existing monitoring service, provided by the cloud provider, or deploy an ad hoc, proprietary tool, to monitor the deployment and runtime characteristics of provisioned instances.

As depicted in Fig. 8.1, the monitoring agent that collects measurements and evaluates if and how much the requirements specified in a request are satisfied. The results of this evaluation are eventually transferred to the cloud manager, a component aimed both at the management of the clouds and at the brokering within the federation. Such information is essential for the cloud manager to be able to build prediction models on the basis of satisfaction of user's requirements. In fact, the degree of satisfaction is the sole input, coming from the monitoring subsystem, exploited by the system to drive its choices. The satisfaction of the users is then paired with the input of user requirement and their associated weights, cloud manager makes a decision on the availability zone that meets the user requirement most. From a more formal perspective, the system is built around two main pillars: *Request attributes* and *Availability Zone Model.*

Request attributes
A user request, for VM placement, is represented by a requirement vector. Let the *ith* user request be represented by the vector, $\mathrm{r_i = r_{i1}, r_{i2}, \ldots, r_{iJ}}$ where $\mathrm{r_{ij}}$, $\mathrm{j} = 1, \ldots, \mathrm{J}$, specifies the jth requirement of user *i* that is expected to be satisfied by the selected availability zone. User requirements may include:

- resources, as the resource amounts required by the user (e.g., CPU, memory, etc.);
- QoS criteria, as quality of service objective that a user wants to achieve (e.g., highest reliability, minimum latency, etc.);
- constraints, as restrictions on possible service provisioning (e.g., locality, throughput, etc.);
- user instance types, as the type of instance the user wants to run;
- user machine types, as the type of machine that the user requires the availability zone to provide.

Availability Zone Model
Availability zones differ in features depending on their actual composition of resources. Attributes such as availability zone size, hardware mix, or management stack (including instance placement policies) result in different levels of reliability and performance. Attribute values that influence the QoS offered by an availability zone for a particular instance type are not known. Generally, QoS data for any particular instance type in a given availability zone are not known a priori. It is, however, possible to measure the QoS parameters after an instance has been deployed. Such measurements may be evaluated against the requirements specified in the request. Unuvar and the other authors of this approach refer to such an evaluation as requirement satisfaction level.

Let $c_{ij} \in [0, 1]$ denote the satisfaction level of requirement r_{ij}. If the requirement r_{ij} is fully satisfied, then $c_{ij} = 1$, otherwise $0 < c_{ij} < 1$. Given such an evaluation, that can be performed using any method able to produce the vector of satisfaction levels, $CT_i = [c_{i1}, c_{i2}, \ldots, c_{iJ}]$, for each incoming request r_i, deployed to an availability zone. The input C_i^T is the only input needed from the monitoring agent.

Predictions

After the definition of attributes, availability zones and satisfaction, it is possible to define the process required to build a prediction model [24]. It can be achieved by learning the behaviour of each availability zone based on the historical data of past requests. The prediction model maps the requirement vector of a request, $\mathrm{r_i} = \mathrm{r_{i1}}, \mathrm{r_{i2}}, \ldots, \mathrm{r_{iJ}}$, to a measure describing customer satisfaction, defined by a utility function, $f(\mathrm{r_i}) \in [0, 1]$. The utility function reaches its maximum value (i.e., 1) when there is complete user satisfaction. The value of $f(\mathrm{r_i})$ depends on how much the requirements of an incoming request is satisfied by the availability zone where the request is addressed. The vector of satisfaction levels, $C_i^T = [c_{i1}, c_{i2}, \ldots, c_{iJ}]$, for each incoming request r_i, is observed after the request is deployed. The satisfaction of some requirements may be more crucial than others, therefore the satisfaction level of each requirement may have different significance. The weight vector $W_i^T = [w_{i1}, w_{i2}, \ldots, w_{iJ}]$ denotes the significance levels for requirement attributes for request r_i. The more the value of w_{ij}, the stronger the significance of *jth* requirement is. One possible way of defining the utility function $f(\mathrm{r_i})$ is to take a linear combination of the satisfaction level for each incoming request C_i and the associated weights WT_i multiplied by an indicator function $\phi(r_i)$. The indicator function is used to set the satisfaction level to zero when the request is rejected. Hence, we define the utility function as,

$$f(r_i) = \phi(r_i) \sum_{j=1}^{J} w_{ij} c_{ij} = \phi(r_i) W_i^T C_i, \tag{8.1}$$

where

$$\phi(r_i) = \begin{cases} 0, & \textit{if the request is rejected} \\ \left(\sum_j w_{ij}\right)^{-1}, & \textit{otherwise} \end{cases} \tag{8.2}$$

Note that the selection of $\phi(r_i) = \left(\sum_j w_{ij}\right)^{-1}$, in the case of no rejection, normalizes the weight vector and limits the maximum possible value of $f(r_i)$ to 1.

To clarify the approach here follows an example, let $r_i = [r_{\mathrm{S}}, r_{\mathrm{L}}, \mathrm{r_I}, \mathrm{r_A}, \mathrm{r_{Mh}}]$ be the requirement vector of an incoming request. The request contains requirements related to the size, supported instance type, infrastructure type, and reliability of an availability zone. The description of the requirement attributes is listed below.

- rS: Requested CPU and RAM resources where rS $\in$ {*micro, small, medium, large, xlarge*}
- rL: Level of reliability where rL $\in$ {*low, medium, high*}
- rI: Tolerance to interruption where rI $\in$ [0, 1] rA: Requested instance type where rA $\in$ {*compute intensive, storage, memory intense Instance*}
- rMh: Requested machine type where rM $\in$ {*Intel Xeon-series, AMD Opteron-series*}.

The measured satisfaction for each requirement is captured by vector C_i^T. Let's assume that for an incoming request with a requirement vector r, such that it holds $r_i = [\text{large}, \mathit{medium}, 1, \mathit{compute\ intensive}, \mathit{Intel\ Xeon}]$, the satisfaction vector is observed as; $C_i^T = [0, 1, 0, 1, 1]$, meaning that the size and tolerance to interruption requirements of the incoming request, $r_S = \{\mathit{large}\}$ and $r_I = \{1\}$, are not satisfied while other requirements are fully satisfied. If the associated weight vector is $W_i^T = [0.2, 0.3, 0.3, 0.1, 0.1]$ then the utility function for r$_i$ is computed as,

$$f(r_i) = \begin{cases} \phi(r_i) W_i^T C_i = 0.5 & \mathit{if\ request\ is\ placed} \\ 0 & \mathit{if\ request\ is\ rejected} \end{cases} \tag{8.3}$$

where $\phi(r_i) = 1$ if the request is placed and 0 otherwise. Note that due to the weights associated with each requirement, the *satisfaction level* did not exceed 0.5 when more than half of the requirements are satisfied.

When different availability zones are able to provide the same services, the availability zone which returns the maximum utility value $f(r_i)$ for the incoming request is likely to be able to satisfy the requirements most. As matter of fact, the exact value of the utility function can be computed only after deployment. However, it can be predicted exploiting available information on past utility values of deployed requests. This predicted value drives the selection of availability zones.

To this end, the assignment process works as follow: the incoming requests are placed into the availability zones by using a random selector, leading to a uniform distribution of the requests to the availability zones. Then, satisfaction level vectors are computed and stored in a properly defined history log. For each incoming request *r*, a training table is built, for every zone, based on its weight vector W and the history of satisfaction vectors of random arrivals to each availability zone from the history log. Once the training tables are built for each zone, the associated prediction models are generated by using classical machine learning techniques for the incoming request.

In short, prediction models are built based on the history log and the weight vector of an incoming request for every zone at the time of arrival. If P_n denotes the prediction model for the satisfaction level of an incoming request placed in an availability zone *n*, P_n is trained by sample records or instances characterized by the tuple $((r_i, f_n(r_i)) : i \in [1, \ldots, I]$, where I is the size of the training set. $f_n(r_i)$ is the empirical value of the utility function in the availability zone *n*, associated with the requirement vector r_i in the history log and new coming request *r*'s weight vector W^T, hence prior requests' satisfaction levels are combined with requests' weights inside the training table. Classification models assume that utility values are discrete. In the case that the utility value is continuous, regression models should be used for prediction.

The training dataset used by P_n to learn the behaviour of availability zone n, composed by I entries, defines for each item a set of J attributes and a target $f_n(r_i)$. After the training phase, P_n learns how to predict the user satisfaction level for the requirement vector r as given by, $P_n(r) = f_n^{\sim}(r)$ where $f_n^{\sim}(r)$ is the predicted satisfaction level for zone n. The estimated mean square prediction error is given by

$$e(P_n) = \frac{1}{I+m} \sum_{l=1}^{I+m} \left[f_n(r_l) - f_n^{\sim}(r_l)\right]^2 \tag{8.4}$$

where m is the number of requests that are deployed to a zone after prediction. In order to find an unbiased estimate of the predicted error, the trained model is validated by testing it against a set of requirement vectors that are not part of the training set. Once each prediction models P_n is generated for each zone, then it is employed to select the availability zone that maximizes the satisfaction of an incoming request.

If the value of the utility function was completely known before deployment, the selection of an availability zone that satisfies the customer requirements would be an obvious choice. However, the utility values, strongly depend on unpublished zone properties and they are not publicly available. Regardless, it is possible to predict them by exploiting machine learning techniques and the historical data as aforementioned and use predicted utility values for availability zone selection.

One possible selection policy uses the maximum predicted utility value to select the availability zone. Hence, if there are n availability zones, the zone that satisfies the requirement vector of the incoming request r most is represented with the Eq. 5.

$$P_{S(r)} = \max_n \{P_{n(r)}\} \forall n \le N \tag{8.5}$$

Here the utility function is maximized when $n = S$, meaning the predicted satisfaction indicates all the user requirements satisfied. Therefore, the zone selector (the cloud broker) assigns zone S as the optimal zone for the incoming request r if that zone S has enough capacity. If the best zone that maximizes the utility function cannot accommodate the request because of resource shortage in zone capacity, then the second-best zone is selected. This procedure repeats until the request is actually placed (or rejected).

Due to the dynamic nature of the cloud environment (i.e. availability, load, time etc.), the accuracy of the prediction models may (and actually does) decrease over time. Clearly, prediction accuracy gets worst when the availability zone features change significantly. These changes may not be publicized or become available to the placement policy. Therefore, as the zone features change, the prediction models need to be retrained to learn the new zone behaviour. **To mitigate the impact of this issue, after each placement, the training data set in the history log can be**

updated. Thus, the prediction models are retrained when the average prediction error increases above a pre-defined threshold. New training samples are collected dynamically to retrain the prediction models when the prediction error exceeds the threshold value.

8.5.2 Machine-Learning Solutions to Deal with Uncertainty

Brokers are aimed at will most likely choose providers that offers advanced and tailored Quality of Service (QoS) guarantees at affordable prices. Thus, providers need to satisfy QoS requirements coming from users and at the same time hold down provisioning cost for keeping their prices competitive. For this purpose, providers should embody resource allocation techniques that consider heterogeneity and dynamicity, key aspects for characterising multi-purpose cloud infrastructures [25]. Heterogeneity refers to the ecosystem of Cloud applications, that are very different in nature and requirements, ranging from embarrassingly-parallel scientific tasks to interactive web applications. Clustering and classification methods can be exploited for addressing such heterogeneity, in order to find similarities in application requirements for a tailored resource allocation. Also, the work of Sharma et al. [26] employs clustering methods for characterising tasks and machines (k-means clustering and simple clustering respectively) by referring to Google data specifically collected for that work. Dynamicity mainly refers to the Cloud considered as a system, whose evolution is difficult to predict because of the best-effort communication model of Internet, the high variability of pattern activation, the faults of infrastructures and so on.

Self-adaptive and self-organising methods can be really effective solutions for addressing such dynamicity. This consideration assumes more relevance when dealing with QoS-aware resource allocations because it often requires optimisation techniques taking into account different goals, which may even be in contrast, one each other (e.g., minimising provisioning costs while maximising application performances). In this context, a promising approach relies on the exploitation of cognitive-based heuristics [27] for choosing, among the available criteria, the optimisation criterion that is more suitable in a particular time instant, without requiring a deep, compete, knowledge on applications as well as on the computing infrastructures. In this way, the system can self-adapt to variation in the underline infrastructure and/or request spikes for a particular class of applications. The concept of cognitive heuristics comes from the adaptive and behavioural psychology science field. Cognitive heuristics are defined as simple and adaptive rules that help the human brain to take decisions in contexts where the selection criterion is unclear, information is partial, time and computational capabilities are limited resources. By exploiting their mathematical description given by cognitive scientists, it is possible to exploit these rules for the definition of efficient algorithms, dealing with decision-making issues. Cognitive heuristics can be exploited in the context of cloud and application brokering for two purposes. Firstly, they allow a cloud resource manager to autonomously and adaptively estimate the risk associated to each selection criteria. Secondly, using the risk estimation, they make possible to combine the costs and risks of the selection options in order to find the best possible resource allocation.

In this context, an interest contribution has been proposed by Anastasi et al. [28] In their work the authors couples cognitive-based models with the Cross-Entropy (CE)-based algorithm [29]. More in detail, the resource allocations, and the linked costs for the system, used as input for the cognitive-based heuristic are selected through the cross-entropy approach. These kinds of algorithms are useful to address any kind of stochastic optimisation problems, as we modelled the resource allocation performed by the Resource Manager presented in this work.

Cross-entropy is exploited to compute resource-application allocations by means of a properly designed and implemented a allocator that enforces the requirements of each application and put a significant effort in optimising the allocations w.r.t. a set of objective functions.

Then, the Cognitive Heuristics is exploited for choosing among different allocations. The allocations computed by the Cross-Entropy based allocator are all valid, meaning both that all of them can be adopted and all respect the requirements of the applications. Each allocation is optimised w.r.t. a specific objective function. The Cognitive Heuristics helps to trade-off between different optimisation directions.

To drive the allocation process, the Resource Manager exploits a set of three objective functions, each one devoted to a specific aim. The three aims shot for by the objective functions are:

Resource Usage Minimisation
This objective function aims at minimising the total amount of resources used for executing the input tasks still satisfying the requirements of the tasks allocated. In other words, we minimise the number of machines of machines used for executing tasks. The idea behind this function is to favour the resource-tasks allocations that minimise the amount of resources that need to be used by a Cloud provider.

Threshold-Balanced Resource Usage
The goal of this objective function is to allocate tasks to machines in a way such that the usage rate of each resource (CPU, RAM, etc.), for each machine exploited in the Cloud, is around a certain value. This means that we minimise the variance of resource amounts dedicated to execute tasks allocated on top of each machine. This objective function aims to guarantee a certain amount of free resources in each machine, which is a simple and handy way to guarantee to applications a certain degree of elasticity without requiring any migration. The elasticity is the ability provided to Cloud applications to exploit additional resources in an on-demand fashion. It is a particularly relevant aspect in the Cloud computing scenario.

Allocation by Task Duration
This function pushes the Resource Manager to allocate tasks in a way such that all the tasks mapped on a certain machine have, more or less, the same duration.

This means that we minimise the variance of time spent by each machine to execute tasks allocated on top of it. The surrounding idea is that if all the tasks associated to a machine spend the same amount of time to be completed, a Cloud provider can easily compute good estimations on the resources availability at a certain time. This objective is particularly relevant in our simplified scenario in which the allocation of resources is performed when a certain number of machines becomes free.

System Architecture

The Architecture of our Resources Allocator is sketched in the (Fig. 8.2). As can be observed, it is composed of three main entities: Cloud Resources, Job Queue and a Resource Manager. According to this model the Resource Manager is activated every time a certain amount of computing resources, among the ones composing the Cloud, became free. The Resource Manager extracts a certain number of tasks from the Queue and tries to allocate them according to a set of objective functions defined by the Cloud Provider.

In the (Fig. 8.3) is shown the zoom-in on the Resource Manager. As can be observed, it is composed of two main subsystems: Cognitive Heuristic module and Cross-Entropy resource allocator. The cognitive heuristic model receives information about available resources and input tasks. By leveraging this information, it performs a number of different invocations of the Cross-Entropy allocator, each with a different objective functions, each returning a different allocation plan.

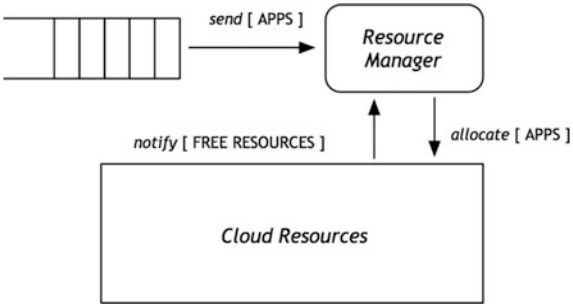


Fig. 8.2 Architectural overview of the system proposed by Anastasi et al. [28]

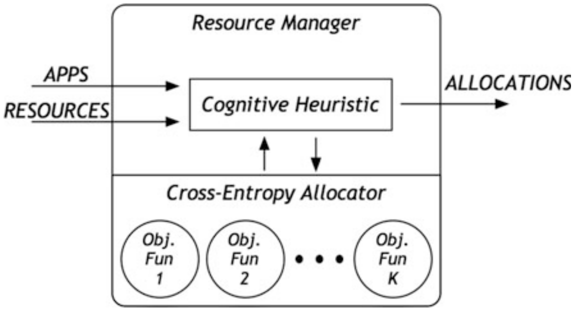


Fig. 8.3 High-level structure of the Resource Manager (Anastasi et al. [28])

The Cognitive Heuristic choose the most promising plan among the proposed ones and enforces it on the Cloud Resources.

An in-depth presentation and discussion of the Cross-entropy technique is beyond the scope and aim of this chapter, so it will not be treated here. Interested reader can refer to the original paper in which it has been proposed, mentioned earlier in this chapter.

In fact, the focus on this section is on the cognitive heuristics, adopted for driving the choice among the different possible solutions identified by means of the cross-entropy solutions. As matter of fact, the Cross-Entropy method allows the Resource Manager to determine a valid solution, meaning an allocation satisfying the resource constraints for each of the objective functions available. The Resource Manager is requested to face the problem of determining the configuration to be chosen, among the proposed ones. In order to make a systematic evaluation among all the potential choices, the Resource Manager proceeds in two steps. Given a configuration computed using a certain objective function, the risk associated to that configuration reflects the degree with which a choice based on the associated objective function could lead to a not optimal configuration. Therefore, at this stage, the problem resolves to be the evaluation between a series of configurations, each characterised by a cost and a measure of how the function generating that configuration could lead to **not optimal** solutions.

The decision-making strategies adopted comes from the cognitive science field. Cognitive-based strategies allow the decision maker to solve complex problems by answering simpler questions. In particular, the Resource Manager needs to autonomously fulfil two tasks. (i) It determines the risks associated with each of the objective functions used to generate configurations, then (ii) it chooses an optimal configuration by taking into account costs and risks.

The relevance of each objective function is estimated in the following way. Resource Manager maintains a memory about previous usages of each objective functions. This memory is modelled according to the ACT-R memory model coming from the cognitive science field [30]. Specifically, for each objective function, the Resource Manager maintains a record where it reports the timestamps of the moments when such function was used to select the best allocation between the ones determined by the Cross-Entropy strategy. Using notions coming from the cognitive sciences, when a new choice has to be taken at a time t, previous usages of a certain objective function are exploited to compute its *memory activation*.

The memory activation value measures the strength with which a given objective function is stored into the Resource Manager memory. In the cognitive sciences, this value can be used in various decision-making processes. In particular, cognitive researchers highlight the fact that the activation of a record in memory corresponds to the log odds of needing that record to achieve a processing goal.

As reported above, in order to make a choice between all the potential alternatives, the Resource Manager has to first evaluate which the risk associated with each of the options. Since the Manager is able to compute, from the memory activation, the probability of needing a given function, we can take, as the risk associated to the same objective function, at time *t,* the probability of not needing that very same function.

With the definitions given so far, we have now to specify how a Resource Manager is able to make the final choice among the proposed allocations, knowing the risks and costs associated to each objective function. In order to achieve this goal, it is leveraged another cognitive-based decision-making strategy, called priority heuristic [31, 32]. Namely, a cognitive model describing the process followed by people to make risky choices. The priority heuristic is a simple decision-making strategy that allows the human brain to deal with situations where choices are characterised by a gain (or, equivalently, a loss or a cost) and a probability to achieve it. From the combination of these attributes, it is not ease to determine which alternative dominate the others. The priority heuristic is a lexicographic heuristic, in the sense that it uses the attributes in a specific order, and each attribute is used only if the previous one was not sufficient to determine the final outcome.

The priority heuristic is used to prune the (potentially large) space of possible choices. At each step, a smaller subset of all the available choices is created, where only the most relevant data is kept, which contains what will be the final result of the evaluation process. Each set created in this way is termed in the cognitive science as a consideration set [33, 34].

Given the set of possible allocations and the sets of the associated costs and risks, the algorithm proceeds to the creation of the first consideration set. This set contains all the allocations whose associated cost is lower than a certain cost threshold. With only one allocation in the consideration set, the priority heuristic algorithm terminates by returning such allocation. Otherwise, the algorithm proceeds by pruning the other allocations.

The second step of the priority heuristic creates a subset of the original consideration containing the allocation associated with the minimum risk value. Other allocations are included in a different consideration set, their associated risks differ quite shortly from the minimum. Again, in case there is only one allocation in the subset, this allocation is considered to be the best choice, and the algorithm terminates.

In case none of the previous two steps was sufficient to choose the best allocation, the last step is used. This step simply selects, as the best option, the allocation associated to the minimum cost, among the ones contained in the second consideration set.

8.5.3 Genetic-Based Solutions for Application Placement

Application placement in clouds often requires the ability to deal with large search spaces, as a consequence of the vast number of requirements and features characterising applications and resources. Genetic Algorithm (GA) is a well-known heuristic approach that permits to iteratively find near-optimal solutions in large search spaces. GA approach it is flexible enough to support multiple constraints and the injection of additional constraints with minimal interventions on the algorithm. This is a crucial feature supporting software reuse in the context of Cloud Computing, where QoS models are continuously enriched as providers constantly extend their systems to support QoS guarantees previously not addressed, such as soft real-time guarantees for virtualised services [35].

In spite of the large amount of benefits that genetic algorithms can brings to cloud brokering and resource management, only a little set of research works focusing on application placement and brokering in cloud are based on such approach.

Essentially, there are three main works proposing the adoption of such paradigm that have been proposed in the scientific literature. From a chronological viewpoint, the first remarkable contribution in this field has been proposed by Zheng et al. [36] who addresses the problem of resource scheduling in Infrastructure as a Service (IaaS) Cloud. They focus on parallel genetic algorithm for speeding the resource allocation process and improving the utilisation of system resources. Even if this is not exactly a broker, it provides a fundamental contribution that is worth to consider in realising efficient brokers targeting large cloud installation.

Another relevant work in the field has been given by Pop et al. [37]. Such approach focuses on the scheduling of independent tasks based on the reputation of resources. Although such is not completely a brokering model, their insight into genetic operators could be leveraged in our work for boosting performances in terms of evolutionary steps for population convergence.

The third work, the last in chronological order, is the one that more closely relate to the core aspects of cloud brokering. In this section, we will mainly focus on this work to show how genetic based solutions can efficiently support cloud brokering. This work is named QBROKAGE, and has been proposed by Anastasi et al. [38].

The brokering algorithm proposed in that work follows the canonical GA and uses a pretty straightforward way to represent applications and resources. In fact, in QBROKAGE, each application is represented by an undirected graph $\langle \mathrm{G_N}, \mathrm{E_M} \rangle$, where G represents the set of N vertexes and E represents the set of M edges connecting the vertexes. Each vertex $g_i \in G$ embodies a VM, each one potentially providing a specialised service, e.g. one VM provides a firewall and another one provides a back-end database. Each edge $e(i,j) \in E$ represents an undirected communication path connecting vertexes g_i and g_j.

Each provider is modelled as a datacentre p_i composed by a set of hosts, that can run one or more VMs depending on their resource availability. The resources belonging to each datacentre are interconnected by a network characterized by a specific set of features, such as bandwidth, latency and security capabilities. Depending on its performance capabilities each datacentre can run a different number of applications, i.e. set of VMs. Each datacentre is characterized by a set of resources and exposes its limits, i.e. the maximum amount of resources that can be assigned to a VM in order to be run on that provider.

To catch the heterogeneity of current pricing models adopted by cloud providers, QBROKAGE considers both providers adopting a per-resource cost model and providers adopting a per-VM cost model. In the per-VM case, cloud providers are modelled as capable of running predefined type of VM provided by customers and charging them for the whole VM used over time. In the per-resource case, we model cloud providers as capable of running each type of VM provided by customers (up to the datacentre limits) and charging them per unit of used resources over time.

QoS attributes are also considered and represented in vectors $Q = \{q^1, \ldots, q^S\}$. In this case the approach followed in QBROKAGE is a quite diffused one, in which attributes are classified in three categories, as defined in the work of Ye et al. [39]:

- **asceding** QoS attributes, in which higher values are better than lower ones;
- **desceding** QoS attributes, in which lower values are better than higher ones;
- **equal** QoS attributes, in which only equality or inequality is meaningful.

When these attributes are used for characterizing the application, they turn into constraint values to be respected by the infrastructure. Thus, for each attribute is indicated as γ_i^j the constraint value related to VM g_i for the QoS attribute q_j. Instead, is denoted as β_i^j the actual value considered for the cloud provider p_i and related to the same attribute q_j.

For checking the adherence of constraint values coming from the application to actual values coming from the infrastructure, a set of inequality constraints $QC = \{QC^1, \ldots, QC^S\}$ is built, whose cardinality is the same of set Q. In particular, there are three possible cases:

- $QC^j = \gamma^j - \beta^j \leq 0$ in case of ascending attributes
- $QC^j = \beta^j - \gamma^j \leq 0$ in case of descending attributes
- $QC^j = \left|\gamma^j - \beta^j\right| - \in\ \leq 0$ in case of nearly equivalent attributes.

When $QC^j \leq 0$ the constraint is respected, otherwise it is not respected.

Moreover, by design, it holds $-1 \leq QC^j \leq 1$ as β^j and γ^j are normalized values with respect to the maximum value for q^j (computed among the set of all providers).

Also, a weight ω^j is associated to each attribute q^j for modelling a search guided by user preferences on particular policies for some attributes. In fact, customers can specify an input vector $V = (v^1, \ldots, v^S)$ of relative values and each ω^j is calculated by equation $\omega^j = \frac{v^j}{|V|}$ where|V| is the unitary norm of vector V.

In QBROKAGE, each solution (chromosome) c is a vector of length N, representing the allocation mapping of each appliance, i.e. a single VM, to a cloud provider. In other words, if $c(i) = j$ then the appliance g_i will be allocated on provider j. As an example, consider the following (Fig. 8.4), where is shown a chromosome representation where VM g_2 is allocated on provider *P1* and thus $c(2) = 1$.

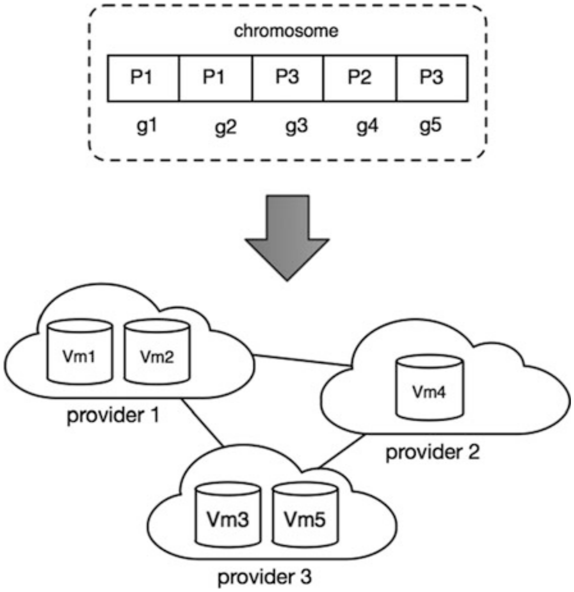


Fig. 8.4 Chromosome driving the brokering process

Algorithm 1: The QBROKAGE approach

```
1. creating_initial_population();
2. while (!termination_condition){
     a. population_evaluation();
     b. selection_for_mating();
     c. one-point_crossover();
     d. random_mutation();
     e. elitism_selection_for_new_generation();
     f. termination_condition_eval();
     }
```

The initial population of S_P individuals (i.e., the potential associations VM-provider) is randomly generated and each individual is evaluated by considering better individuals with those having higher fitness values. The population selected for mating, is randomly chosen with a rate R_{cross} among the total and the crossover strategy is the random one-point crossover. Thus, a number of $S_P \times R_{cross}$ crossover operations are performed for each generation and each operation produces 2 individuals. Then, mutation is applied with a probability of P_{mut} to each gene in each individual. After applying mutation, an elitist selector is used to select the top 90% of the population size, whilst the remaining is obtained by cloning the best selected up to reach the population size. Clearly, this is just the illustration of the principle, without dealing with numeric values, which can be found in the original paper, along with a complete discussion of the achievement of the actual contribute.

When evaluating a solution, the algorithm of Anastasi et al., here reported, first considers each gene g by defining a column vector $\mathrm{D_g} = \left(\mathrm{d_g^1}, \ldots, \mathrm{d_g^S}\right)$ as the distance of an allele from constraint satisfaction. For each QoS constraint in the set $\left\{\mathrm{q^1}, \ldots, \mathrm{q^S}\right\}$ such a distance d_g is computed by using inequalities aforementioned.

According to the problem formulation, we consider values $-1 \leq QC_g^j < 0$ as those satisfying the constraint j for gene g, with better solutions in minimising the gradient direction. We recall that each QoS attribute j has associated a weight ω_j.

It is quite common to model fitness functions as maximisation problems, the very same happens in QBROKAGE, where the authors define the fitness of a gene g as $l(g) = AD_g$.

where A is the square matrix $A = \begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{s,s} \end{pmatrix}$

with each element in the diagonal defined as $a_{j,j} = \begin{cases} -\omega^j & if\, QC^j < 0 \\ 0 & if\, QC^j > 0 \end{cases}$

Then, the fitness function of each chromosome c is defined as $F(c) = \sum_{i=1}^{N} l(g_i) \times K$ with N being equal to the number of genes of c and K being equal to a constant defined for awarding those chromosomes with a high number of genes that correspond to allocations respecting constraints.

8.6 Conclusion

This chapter present a bird-eye-view on the current status of works in the literature dealing with cloud brokering and machine learning and intelligent systems from a broader viewpoint. The chapter shortly introduce the aims of machine learning and give to the reader a briefly summarisation of the existing approaches to machine learning.

The chapter then discusses the connection between machine learning and cloud brokering, introducing the current landscape of the existing approaches.

The presentation is given by focusing on the most relevant solutions existing in the scientific literature, grouped on three main sub-topics: machine learning for application placement in cloud federation, machine learning in the brokering of application characterised by high degrees of uncertainty and machine based models exploiting evolutionary computing to achieve better performances.

References

1. S. Phil, *Too Big to Ignore: The Business Case for Big Data* (Wiley, 2013), p. 89. ISBN 978-1-118-63817-0
2. M.N. Wernick, Y. Yang, J.G. Brankov, G. Yourganov, S.C. Strother, Machine Learning in Medical Imaging, IEEE Sig. Process. Mag. **27**(4), 25–38 (2010)
3. H. Mannila, *Data Mining: Machine Learning, Statistics, And Databases.* International Conference. Scientific and Statistical Database Management IEEE Computer Society 1996
4. S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd edn. (Prentice Hall, New Jersey, 1995). ISBN 978-0137903955 (2003)
5. G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules, in *Knowledge Discovery in Databases*, ed. by Gregory Piatetsky-Shapiro, William J. Frawley (AAAI/MIT Press, Cambridge, 1991)
6. R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data—SIGMOD'93* (1993), p. 207
7. S. Haykin, *Neural Networks: A Comprehensive Foundation* (Prentice Hall, 1999). ISBN 0-13-273350-1
8. W. Buntine, Theory refinement on Bayesian networks, eds. by B.D.D' Ambrosio, P. Smets, P.P. Bonissone, in *Proceedings of the Seventh Annual Conference on Uncertainty Artificial Intelligence*, (Morgan Kaufmann, San Francisco, 1991), pp. 52–60
9. B. Everitt, *Cluster Analysis*, (Wiley, Chichester). ISBN 9780470749913
10. J.R. Quinlan, Induction of decision trees. Mach. Learn. 1, pp. 81–106 (Kluwer Academic Publishers, Dordrecht, 1986)
11. G. Ian, B. Yoshua, C. Aaron, *Deep Learning* (MIT Press, Cambridge, 2016)
12. Y. Bengio, Learning deep architectures for AI (pdf). Found. Trends. Mach. Learn **2**(1), 1–127 (2009)
13. Y. Bengio, Y. LeCun, G. Hinton, Deep Learning. Nature **521**, 436–444 (2015)
14. L. De Raedt, *A Perspective on Inductive Logic Programming. The Workshop on Current and Future Trends in Logic Programming* (Springer LNCS, Shakertown, 1999)
15. Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives. IEEE Trans. PAMI, Spec. Issue. Learn. Deep. Archit. **35**, 1798–1828 (2013)
16. B. Kulis, Metric learning: a survey. Found. Trends. Mach. Learn. (2012)
17. W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Section 16.5. Support Vector Machines. *Numerical Recipes: The Art of Scientific Computing,* 3rd edn. (Cambridge University Press, New York, 2007). ISBN 978-0-521-88068-8
18. M. Melanie, *An Introduction to Genetic Algorithms* (MIT Press, 1996). ISBN 9780262133166
19. S. M. Weiss, N. Indurkhya, Rule-based machine learning methods for functional prediction. J. Artif. Intell. Res. 3 (1995)
20. R. J. Urbanowicz,J. H. Moore, Learning classifier systems: a complete introduction, review, and roadmap. J. Artif. Evol. Appl. 1–25 (2009)
21. A. Celesti, F. Tusa, M. Villari, A. Puliafito, in How to enhance cloud architectures to enable cross-federation. *IEEE 3rd International Conference on Cloud Computing*, (Miami, 2010), pp. 337–345. doi:10.1109/CLOUD.2010.46
22. E. Carlini, M. Coppola, P. Dazzi, L. Ricci, G. Righetti Cloud federations in contrail, in *Proceedings of Euro-Par 2011: Parallel Processing Workshops,* **7155**, ed. by Alexander Mea (Springer, Heidelberg, 2012), pp. 159–168
23. B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muñoz, G. Toffetti, Reservoir - When one cloud is not enough. Computer **44**(3), 44–51 (2011)
24. L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, *Classification and Regression Trees* (CRC Press, New York, 1999)

25. C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, *Heterogeneity and dynamicity of clouds at scale: Google trace analysis, in ACM Symposium on Cloud Computing (SoCC)* (CA, USA, San Jose, 2012)
26. B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, C. R. Das, Modeling and synthesizing task placement constraints in google compute clusters, *in Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 3
27. E. Brandstatter, G. Gigerenzer, R. Hertwig, The priority heuristic: making choices without trade-offs. Psychol. Rev. **113**(2), 409 (2006)
28. G. F. Anastasi, P. Cassarà, P. Dazzi, A. Gotta, M. Mordacchini and A. Passarella, A Hybrid Cross-Entropy Cognitive-Based Algorithm for Resource Allocation in Cloud Environments, *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*, (London, 2014), pp. 11–20. doi:10.1109/SASO.2014.13
29. R.Y. Rubinstein, D.P. Kroese, *The Cross-Entropy Method: A Unified Approach To Combinatorial Optimization, Monte–Carlo Simulation And Machine Learning* (Springer, Berlin, 2004)
30. J.R. Anderson, C. Lebiere, *The Atomic Components Of Thought* (Lawrence Erlbaum Inc., Publisher, 1998)
31. M. Drechsler, K. Katsikopoulos, G. Gigerenzer, Axiomatizing bounded rationality: the priority heuristic, Theor. Decis. 1–14 (2011)
32. E. Brandstatter, M. Gussmack, The cognitive processes underlying risky choice. Journal of Behavioral Decision Making **26**(2), 185–197 (2013)
33. M. Conti, M. Mordacchini, A. Passarella, Design and performance evaluation of data dissemination systems for opportunistic networks based on cognitive heuristics, ACM Trans. Auton. Adapt. Syst. 8(3), 12:1–12:32 (2013)
34. J.N. Marewski, W. Gaissmaier, L.J. Schooler, D.G. Goldstein, G. Gigerenzer, From recognition to decisions: extending and testing recognition-based models for multialternative inference. Psychon. Bull. Rev. **17**(3), 287–309 (2010)
35. T. Cucinotta, G. Anastasi, L. Abeni, *Respecting Temporal Constraints in Virtualised Services, in Computer Software and Applications Conference. COMPSAC'09. 33rd Annual IEEE International*, vol. 2, pp. 73–78 2009
36. Z. Zheng, R. Wang, H. Zhong, X. Zhang, *An approach for cloud resource scheduling based on Parallel Genetic Algorithm*, in *2011 3rd International Conference on Computer Research and Development (ICCRD)*, vol. 2, pp. 444–447 2011
37. F. Pop, V. Cristea, N. Bessis, and S. Sotiriadis, Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments, in *Proceedings of the 2013 27th International Conference on Advanced Information Networking and Applications Workshops, ser. WAINA'13.*, (IEEE Computer Society, Washington, 2013), pp. 772–776
38. G. F. Anastasi, E. Carlini, M. Coppola and P. Dazzi, QBROKAGE: A Genetic Approach for QoS Cloud Brokering, *2014 IEEE 7th International Conference on Cloud Computing*, Anchorage, AK, 2014, pp. 304–311. doi:10.1109/CLOUD.2014.49
39. Z. Ye, X. Zhou, A. Bouguettaya, *Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing,* eds by J. Yu, M. Kim, R. Unland In Database Systems for Advanced Applications, ser. Lecture Notes in Computer Science, Vol 6588 (Springer, Heidelberg, 2011) pp. 321–334