Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Online optimization for scheduling preemptable tasks on IaaS cloud systems

Jiayin Li[a], Meikang Qiu[a], Zhong Ming[b,*], Gang Quan[c], Xiao Qin[d], Zonghua Gu[e]

[a] Department of Elec. and Comp. Engr., University of Kentucky, Lexington, KY 40506, USA
[b] College of Computer Science and Software, Shenzhen University, Shenzhen 518060, China
[c] College of Engr. and Comp., Florida International University, Miami, FL 33174, USA
[d] Department of Comp. Sci. and Software Engr., Auburn University, Auburn, AL 36849, USA
[e] College of Computer Science, Zhejiang University, Hangzhou 310027, China

ARTICLE INFO

Article history:
Received 22 January 2011
Received in revised form
26 January 2012
Accepted 1 February 2012
Available online 16 February 2012

Keywords:
Cloud computing
Online scheduling
Feedback
Preemptable scheduling

ABSTRACT

In *Infrastructure-as-a-Service* (IaaS) cloud computing, computational resources are provided to remote users in the form of leases. For a cloud user, he/she can request multiple cloud services simultaneously. In this case, parallel processing in the cloud system can improve the performance. When applying parallel processing in cloud computing, it is necessary to implement a mechanism to allocate resource and schedule the execution order of tasks. Furthermore, a resource optimization mechanism with preemptable task execution can increase the utilization of clouds. In this paper, we propose two online dynamic resource allocation algorithms for the IaaS cloud system with preemptable tasks. Our algorithms adjust the resource allocation dynamically based on the updated information of the actual task executions. And the experimental results show that our algorithms can significantly improve the performance in the situation where resource contention is fierce.

1. Introduction

In cloud computing, a cloud is a cluster of distributed computers providing on-demand computational resources or services to the remote users over a network [14]. In an Infrastructure-as-a-Service (IaaS) cloud, resources or services are provided to users in the form of leases. The users can control the resources safely, thanks to the free and efficient virtualization solutions, e.g., the Xen hypervisor [37]. One of the advantages of the IaaS clouds is that the computational capacities providing to end-users are flexible and efficient. The *virtual machines* (VMs) in Amazon's Elastic Compute Cloud are leased to users at the price of ten cents per hour. Each VM offers an approximate computational power of a 1.2 GHz Opteron processor, with 1.7 GB memory and 160 GB disk space. For example, when a user needs to maintain a database with a certain disk space for a month, he/she can rent a number of VMs from the cloud, and return them after that month. In this case, the user can minimize the costs. And the user can add or remove resources from the cloud to meet peak or fluctuating service demands and pay only the capacity used.

Cloud computing is emerging with growing popularity and adoption [1]. However, there is no data center that has unlimited capacity. Thus, in case of significant client demands, it may be necessary to overflow some workloads to another data center [11]. These workload sharing can even occur between private and public clouds, or among private clouds or public clouds. The workload sharing is able to enlarge the resource pool and provide even more flexible and cheaper resources. To collaborate the execution across multiple clouds, the monitoring and management mechanism is a key component and requires the consideration of provisioning, scheduling, monitoring, and failure management [11]. Traditional monitoring and management mechanisms are designed for enterprise environments, especially a unified environment. However, the large scale, heterogeneous resource provisioning places serious challenges for the management and monitoring mechanism in multiple data centers. For example, the Open Cirrus, a cloud computing testbed, consists of 14 geographically distributed data center in different administrative domains around the world. Each data center manages at least 1000 cores independently [3]. The overall testbed is a heterogeneous federated cloud system. It is important for the monitoring and management mechanism to provide the resource pool, which includes multiple data centers, to clients without forcing them to handle issues, such as the heterogeneity of resources and the distribution of the workload. Virtualization in cloud computing, such as VMs, has been intensively studied recently. However, scheduling workloads across multiple heterogeneous clouds/data centers has not been well studied in the literature. To the best of our knowledge, this is the first paper to address the scheduling issue in the federated heterogeneous multi-cloud system.

* Corresponding author.
E-mail addresses: jli6@engr.uky.edu (J. Li), mqiu@engr.uky.edu, qiumeikang@yahoo.com (M. Qiu), mingz@szu.edu.cn (Z. Ming), Gang.Quan@fiu.edu (G. Quan), xqin@auburn.edu (X. Qin), zgu@zju.edu.cn (Z. Gu).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.02.002

A large number of applications running on cloud systems are those compute on large data corpora [22]. These "big data" applications draw from information source such as digital media collections, virtual worlds, simulation traces, data obtain from scientific instruments, and enterprise business databases. These data hungry applications require scalable computational resources. Fortunately, these applications exhibit extremely good parallelism [22]. Using a "map/reduce" approach in the cloud application development, large batch processes can be partitioned into a set of discrete-linked processes, which we call tasks. These tasks can be executed in parallel to improve response time [4]. In Fedex's data center, a four-hour batch process can be successfully runs in 20 min after the "map/reduce" [4]. When applying parallel processing in executing these tasks, we need to consider the following questions: (1) how to allocate resources to tasks; (2) in what order the clouds should execute tasks, since tasks have data dependencies; (3) how to schedule overheads when VMs prepare, terminate or switch tasks. Resource allocation and scheduling can solve these three problems. Resource allocation and task scheduling have been studied in high performance computing [8, 16] and in embedded systems [30,29]. However, the autonomic feature and the resource heterogeneity within clouds [14] and the VM implementation require different algorithms for resource allocation and task scheduling in the IaaS cloud computing, especially in the federated heterogeneous multi-cloud system.

The two major contributions of this paper are:

- We present a resource optimization mechanism in heterogeneous IaaS federated multi-cloud systems, which enables preemptable task scheduling. This mechanism is suitable for the autonomic feature within clouds and the diversity feature of VMs.
- We propose two online dynamic algorithms for resource allocation and task scheduling. We consider the resource contention in the task scheduling.

In Section 2, we discuss works related to this topic. In Section 3, models for resource allocation and task scheduling in IaaS cloud computing system are presented, followed by an motivation example in Section 4. We propose our algorithms in Section 5, followed by experimental result in Section 6. Finally, we give the conclusion in Section 7.

2. Related works

Cloud system has been drawing intensive research interests in the recent years. A number of public clouds are available for customer and researchers, such as Amazon AWS [38], GoGrid [39], and Rackspace [40]. Some other companies also provide cloud services, such as Microsoft [41], IBM [42], Google [43], and HP [44]. To benefit the cloud research, open source cloud services are under way, such as Eucalyptus [45], Open Nebula [25], Tashi [22], RESEVOIR [46], and Open Cirrus [3]. Open Cirrus is a cloud testbed consists of 14 distributed data centers among the world. Essentially, it is a federated heterogeneous cloud system, which is similar to the target cloud system in our paper.

Data intensive applications are the major type of applications running in the cloud computing platform. Most of the data intensive applications can be modeled by MapReduce programming model [7]. In MapReduce model, user specify the map function that can be executed independently, and the reduce function that gather results from the map function and generate the final result. The runtime system automatically parallelizes the map function and distributes them in the cloud system. Apache Hadoop is a popular framework, inspired by MapReduce, for running the data-intensive application in IaaS cloud systems [15]. Both reliability and data motion are transparently provided in Hadoop framework. MapReduce programming model and Hadoop distributed file system are implemented in the open-source Hadoop framework. All-pairs, an high level abstraction, was proposed to allow the easy expression and efficient execution of data intensive applications [26]. Liu et al. designed a programming model, GridBatch, for large scale data intensive batch applications [23]. In GridBatch, user can specify the data partitioning and the computation task distribution, while the complexity of parallel programming is hidden. A dynamic split model was designed to enhance the resource utilization in MapReduce platforms [48]. A priority-based resource allocation approach as well as a resource usage pipeline are implemented in this dynamic split model. Various scheduling methods for data-intensive services were evaluated [6], with both soft and hard *service level agreements* (SLA). However, the problem of scheduling workloads in heterogeneous multi-cloud platform was not considered in the related work mentioned above.

Virtualization is an important part in cloud computing. Emeneker et al. propose an image caching mechanism to reduce the overhead of loading disk image in virtual machines [9]. Fallenbeck et al. present a dynamic approach to create virtual clusters to deal with the conflict between parallel and serial jobs [10]. In this approach, the job load is adjusted automatically without running time prediction. A suspend/resume mechanism is used to improve utilization of physical resource [36]. The overhead of suspending/resume is modeled and scheduled explicitly. But the VM model considered in [36] is homogeneous, so the scheduling algorithm is not applicable in heterogeneous VMs models.

Computational resource management in cloud computing has been studied in the literature recently. To make resource easy for users to manage collectively, CloudNet [50] provides virtual private clouds from enterprise machines and allocates them via public clouds. Computation-intensive users can reserve resources with on-demand characteristics to create their virtual private clouds [2,12,5,47,21,19]. However, CloudNet focuses on providing secure links to cloud for enterprise users, resource allocation is not an objective in CloudNet. Lease-based architecture [19,35] is widely used in reserving resource for cloud users. In [19], applications can reserve group of resources using leases and tickets from multiple sites. Haizea [35] supports both the best-effort and the advanced reservation leases. The priorities of these two kinds of leases are different. The utilization of the whole system is improved. The model of job in these two paper is a batch job model, which mean every application is scheduled as independent. Data dependencies are not considered. Thus this method cannot be "map/reduce" and parallelized among multiple data centers. In our proposed resource allocation mechanism, we model the data dependencies among an application, and distribute the application among multiple data centers at the task level, leading to more flexible and more efficient resource allocation schedules.

Wilde et al. proposed Swift, a scripting language for distributed computing [49]. Swift focuses on the concurrent execution, composition, and coordination of large scale independent computational tasks. A workload balancing mechanism with adaptive scheduling algorithms is implemented in Swift, based on the availability of resources. A dynamic scoring system is designed to provide an empirically measured estimate of a site's ability to bear load, which is similar to the feedback information mechanism proposed in our design. However, the score in the Swift is decreased only when the site fails to execute the job. Our approach has a different use of the feedback information. The dynamic estimated finish time of remote site is based on the previous executions on this site in our approach. Therefore, even a "delayed but successful" finish of a job leads to a longer estimated finish time in the next run in our approach. ReSS is used in the Swift as the resource selection service [13]. ReSS requires a central information repository to gather information from different nodes or clusters. However, our approach is a decentralized approach that does not need any central information repository.

A system that can automatically scale its share of infrastructure resources is designed in [33]. The adaptation manager monitors and autonomically allocating resources to users in a dynamic way, which is similar to the manager server in our proposed mechanism. However, this centralized approach cannot fit in the future multi-provider cloud environment, since different providers may not want to be controlled by such a centralized manager. Another resource sharing system that can trade machines in different domains without infringing autonomy of them is developed in [32]. A machine broker of a data center is proposed to trade machines with other data centers, which is a distributed approach to share resource among multiple data centers. However, the optimization of resource allocation is not considered in this paper. Our proposed resource allocation mechanism is a distributed approach. A manager server of a cloud communicates with others, and shares workloads with our dynamic scheduling algorithm. Our approach can improve federated heterogeneous cloud systems. Moreover, it can be adapted in the future multi-provider cloud system.

3. Model and background

3.1. Cloud system

In this paper, we consider an infrastructure-as-a-service (IaaS) cloud system. In this kind of system, a number of data center participates in a federated approach. These data centers deliver basic on-demand storage and compute capacities over Internet. The provision of these computational resources is in the form of virtual machines (VMs) deployed in the data center. These resources within a data center form a cloud. Virtual machine is an abstract unit of storage and compute capacities provided in a cloud. Without loss of generality, we assume that VMs from different clouds are offered in different types, each of which has different characteristics. For example, they may have different numbers of CPUs, amounts of memory and network bandwidths. As well, the computational characteristics of different CPU may not be the same.

For a federated cloud system, a centralized management approach, in which a super node schedule tasks among multiple clouds, may be an easy way to address the scheduling issues in such system. However, as authors in [3,11] have indicated, the future cloud computing will consist of multiple cloud providers. In this case, the centralized management approach may be accepted by different cloud providers. Thus we propose a distributed resource allocation mechanism that can be used in both federated cloud system or the future cloud system with multiple providers.

As shown in Fig. 1, in our proposed cloud resource allocation mechanism, every data center has a manager server that knows the current statuses of VMs in it own cloud. And manager servers communicate with each other. Clients submit their tasks to the cloud where the dataset is stored. Once a cloud receives tasks, its manager server can communicate with manager servers of other clouds, and distribute its tasks across the whole cloud system by assigning them to other clouds or executing them by itself.

When distributing tasks in the cloud system, manager servers should be aware of the resource availabilities in other clouds, since there is not a centralized super node in the system. Therefore, we need the resource monitoring infrastructure in our resource allocation mechanism. In cloud systems, resource monitoring infrastructure involves both producers and consumers. Producers generate status of monitored resources. And consumers make use of the status information [17]. Two basic messaging methods are used in the resource monitoring between consumers and producers: the pull mode and the push model [51]. Consumers pull information from producers to inquire the status in the pull mode. In the push mode, when producers update any resource status, they push the information to the consumers. The advantage of the push mode is that the accuracy is higher when the threshold of a status update, i.e., trigger condition, is defined properly. And the advantage of the pull mode is that the transmission cost is less when the inquire interval is proper [17].

In our proposed cloud system resource allocation mechanism, we combine both communication modes in the resource monitoring infrastructure. In our proposed mechanism, when the manager server of cloud A assigns an application to another cloud B, the manager server of A is the consumer. And the manager server of B is the producer. manager server of A needs to know the resource status from the manager server of B in two scenarios: (1) when the manager server of A is considering assigning tasks to cloud B, the current resource status of cloud B should be taken into consideration. (2) When there is an task is assigned to cloud B by manager server of A, and this task is finished, manager server of A should be informed.

We combine the pull and the push mode as the following:

- A consumer will pull information about the resource status from other clouds, when it is making scheduling decisions.
- After an application is assigned to another cloud, the consumer will no longer pull information regarding to this application.
- When the application is finished by the producer, the producer will push its information to the consumer. The producer will not push any information to the consumer before the application is finished.

In a pull operation, the trigger manager server sends a task check inquire to manager servers of other clouds. Since different cloud providers may not be willing to share detailed information about their resource availability, we propose that the reply of a task check inquire should be as simple as possible. Therefore, in our proposed resource monitoring infrastructure, these target manager servers give only responses at the earliest available time of required resources, based on its current status of resources. And no guarantee or reservation is made. Before target manager servers check their resource availability, they first check the required dataset locality. If the required dataset is not available in their data center, the estimated transferring time of the dataset from the trigger cloud will be included in the estimation of the earliest available time of required resources. Assuming the speed of transferring data between two data centers is S_c, and the size of the required dataset is M_S, then the preparation overhead is M_S/S_c. Therefore, when a target cloud already has the required dataset in its data center, it is more likely that it can respond sooner at the earliest available time of required resources, which may lead to an assignment to this target cloud. In a push operation, when B is the producer and A is the consumer, the manager server of B will inform the manager server of A the time when the application is finished.

When a client submits his/her workload, typically an application, to a cloud, the manager server first partitions the application into several tasks, as shown in Fig. 2. Then for each task, the manager server decides which cloud will execute this task based on the information from all other manager servers and the data dependencies among tasks. If the manager server assigns a task to its own cloud, it will store the task in a queue. And when the resources and the data are ready, this task is executed. If the manager server of cloud A assigns a task to cloud B, the manager server of B first checks whether its resource availabilities can meet the requirement of this task. If so, the task will enter a queue waiting for execution. Otherwise, the manager server of B will reject the task.

Before a task in the queue of a manager server is about to be executed, the manager server transfers a disk image to all the computing nodes that provide enough VMs for task execution.

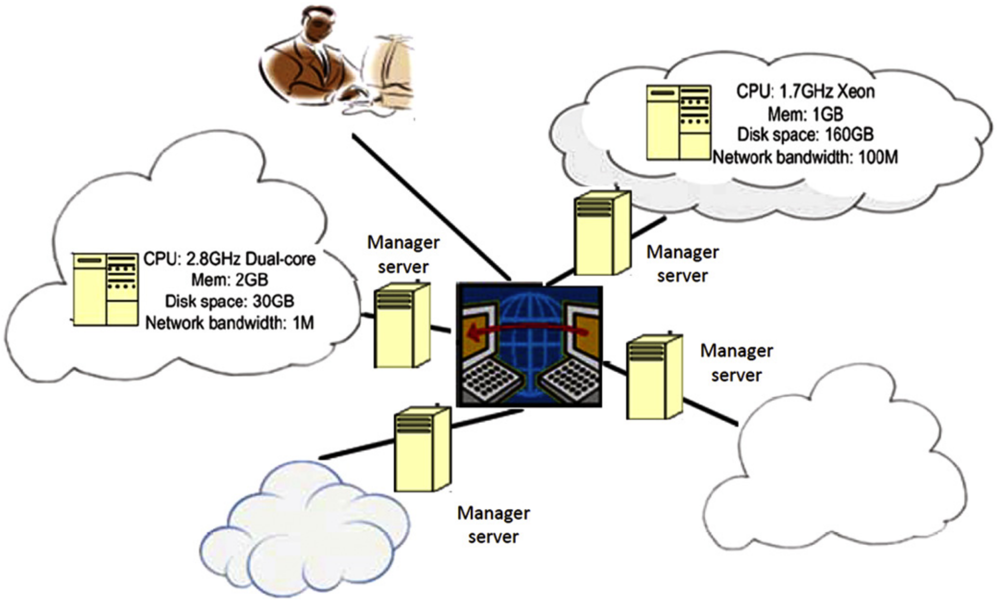


Fig. 1. An example of our proposed cloud resource allocation mechanism. Heterogeneous VMs are provided by multiple clouds. And clouds are connected to the Internet via manager servers.

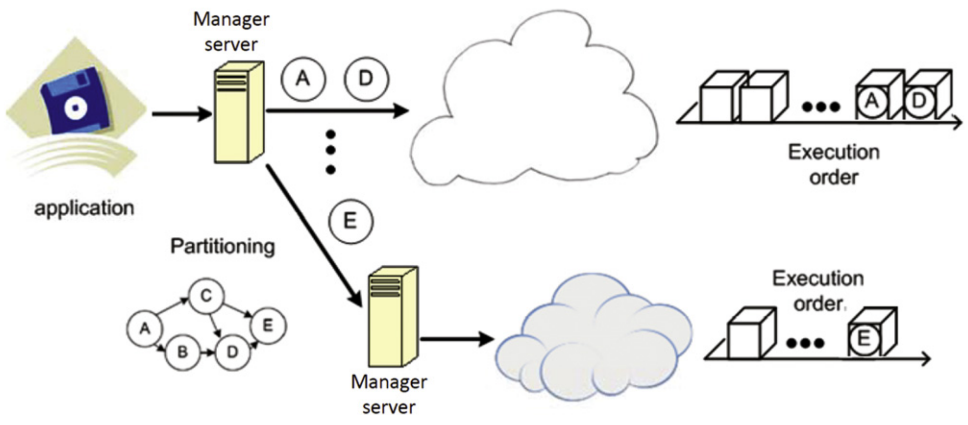


Fig. 2. When an application is submitted to the cloud system, it is partitioned, assigned, scheduled, and executed in the cloud system.

We assume that all required disk images are stored in the data center and can be transferred to any clouds as needed. We use the multicasting to transfer the image to all computing nodes within the data center. Assuming the size of this disk image is S_I, we model the transfer time as S_I/b, where b is the network bandwidth. When a VM finishes its part of the task, the disk image is discarded from computing nodes.

3.2. Resource allocation model

In cloud computing, there are two different modes of renting the computing capacities from a cloud provider.

- Advance Reservation (AR): Resources are reserved in advance. They should be available at a specific time.
- Best-effort: Resources are provisioned as soon as possible. Requests are placed in a queue.

A lease of resource is implemented as a set of VMs. And the allocated resources of a lease can be described by a tuple (n, m, d, b), where n is number of CPUs, m is memory in megabytes, d is disk space in megabytes, and b is the network bandwidth in megabytes per second. For the AR mode, the lease also includes the required start time and the required execution time. For the best-effort and the immediate modes, the lease has information about how long the execution lasts, but not the start time of execution. The best-effort mode is supported by most of the current cloud computing platform. The Haizea, which is a resource lease manager for OpenNebula, supports the AR mode [37]. The "map" function of "map/reduce" data-intensive applications are usually independent. Therefore, it naturally fits in the best-effort mode. However, some large scale "reduce" processes of data-intensive applications may needs multiple reducers. For example, a simple "wordcount" application with tens of PBs of data may need a parallel "reduce" process, in which multiple reducers combine the results of multiple mappers in parallel. Assuming there are N reducers, in the first round of parallel "reduce", each of N reducers counts $1/N$ results from the mappers. Then $N/2$ reducers receive results from the other $N/2$ reducers, and counts $2/N$ results from the last round of reducing. It repeats $\log_2 N + 1$ rounds. Between two rounds, reducers need to communicate with others. Therefore, an AR mode is more suitable for these data-intensive applications.

When supporting the AR tasks, it may leads to a utilization problem, where the average task waiting time is long, and machine utilization rate is low. Combining AR and best-effort in a preemptable fashion can overcome this problems [35]. In this paper, we assume that a few of applications submitted in the cloud system are in the AR mode, while the rest of the applications are in the best-effort mode. And the applications in AR mode have higher priorities, and are able to preempt the executions of the best-effort applications.

When an AR task A needs to preempt a best-effort task B, the VMs have to suspend task B and restore the current disk image of task B in a specific disk space before the manager server transfers the disk image of tasks A to the VMs. When the task A finishes, the VMs will resume the execution of task B. We assume that there is a specific disk space in every node for storing the disk image of suspended task.

There are two kinds of AR tasks: one requires a start time in future, which is referred to as "non-zero advance notice" AR task; the other one requires to be executed as soon as possible with higher priority than the best-effort task, which is referred to as "zero advance notice" AR task. For a "zero advance notice" AR task, it will start right after the manager server makes the scheduling decision and assign it a cloud. Since our scheduling algorithms, mentioned in Section 5, are heuristic approaches, this waiting time is negligible, compared to the execution time of task running in the cloud system.

3.3. Local mapping and energy consumption

From the user's point of view, the resources in the cloud system are leased to them in the term of VMs. Meanwhile, from the cloud administrator's point of view, the resources in the cloud system is utilized in the term of servers. A server can provide the resources of multiple VMs, and can be utilized by several tasks at the same time. One important function of the manager server of each cloud is to schedule its tasks to its server, according to the number of required VMs. Assuming there are a set of tasks T to schedule on a server S, we define the remaining workload capacity of a server S is $C(S)$, and the number of required VM by task t_i is $wl(t_i)$. The server can execute all the tasks in T only if:

$$C(S) \geq \sum_{t_i \in T}(wl(t_i)). \tag{1}$$

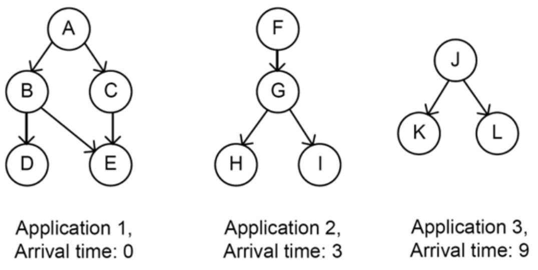
We assume servers in the cloud system work in two different modes: the active mode and the idle mode. When the server is not executing any task, it is switched to the idle mode. When tasks arrive, the server is switched back to the active mode. The server consumes much less energy in the idle mode than that in the active mode.

3.4. Application model

In this paper, we use the *Directed Acyclic Graphs* (DAG) to represent applications. A DAG $T = (V, E)$ consists of a set of vertices V, each of which represents a task in the application, and a set of edges E, showing the dependences among tasks. The edge set E contains edges e_{ij} for each task $v_i \in V$ that task $v_j \in V$ depends on. The weight of a task represents the type of this task. Given an edge e_{ij}, v_i is the immediate predecessor of v_j, and v_j is called the immediate successor of v_i. A task only starts after all its immediate predecessors finish. Tasks with no immediate predecessor are entry-node, and tasks without immediate successors are exit-node.

Although the compute nodes from the same cloud may equip with different hardware, the manager server can treat its cloud as a homogeneous system by using the abstract compute capacity unit and the virtual machine. However, as we assumed, the VMs from different clouds may have different characteristics. So the whole cloud system is a heterogeneous system. In order to describe the difference between VMs' computational characteristics, we use an $M \times N$ execution time matrix (ETM) E to indicate the execution time of M types of tasks running on N types of VMs. For example,

a


A
B
C
D
E
F
G
H
I
J
K
L
Application 1, Arrival time: 0
Application 2, Arrival time: 3
Application 3, Arrival time: 9

b

	A	B	C	D	E	F	G	H	I	J	K	L
Cloud 1	2	6	5	7	5	4	8	2	4	8	9	2
Cloud 2	3	8	3	10	9	2	8	3	5	4	3	3
Cloud 3	5	4	8	5	2	3	4	6	7	6	7	4

c

	A	B	C	D	E	F	G	H	I	J	K	L
RR	1	2	3	1	2	1	2	3	1	1	2	3
Sch	1	3	2	3	3	2	3	1	2	2	2	1

Fig. 3. (a) The DFG of three applications, (b) the execution time table, and (c) two different task assignments, where "RR" is the round-robin approach, and "Sch" is using the list scheduling.

the entry e_{ij} in E indicate the required execution time of task type i when running on VM type j. We also assume that a task requires the same lease (n, m, d, b) no matter on which type of VM the task is about to run.

4. Motivational example

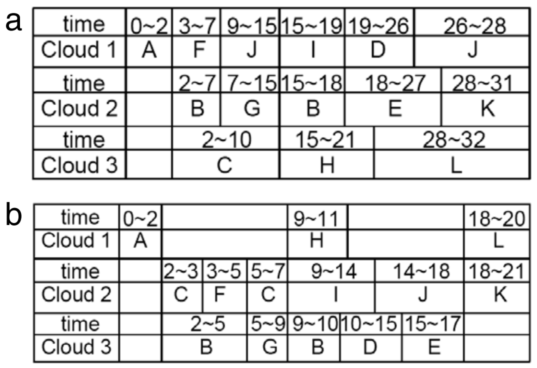
4.1. An example of task scheduling in CMP

First we give an example for resource allocation in a cloud system. We schedule three applications in a three-cloud system. The DFGs representing these applications are shown in Fig. 3(a). Application 1 and 3 are best-effort applications, and Application 2 is AR applications. For simplicity, we assume that every cloud only execute one task at a time, and that the time to load an image of a task is negligible. We will relax these assumptions in the later part of this paper. The execution times (t) of each task in these applications running on different cloud are shown in Fig. 3(b).

4.2. Round-robin vs. list scheduling

The round-robin algorithm is one of the load balancing algorithms used in cloud systems, such as the GoGrid [31]. As shown in the "RR" row of Fig. 3(c), the tasks are assigned to the clouds evenly, regardless of the heterogeneous performance across different clouds. The execution orders of three clouds are presented in Fig. 4(a). In this schedule, task G preempts task B at time 7, since task G is an AR task. And task J is scheduled as soon as possible, starting at time 9, pausing at time 15, and resuming right after previously assigned tasks, i.e., tasks I and D. The total execution time is 32. We assume the execution time of a given application starts from the time when the application is submitted to the time when the application is done. With this scheduling, the average of three application execution time is 22.67 time unit.

By using our CLS algorithm, we generate a schedule with the consideration of the heterogeneous performance in the cloud system. The tasks assignment is shown in the "Sch" row of Fig. 3(c). And the execution order of three clouds are shown in Fig. 4(b). In this schedule, tasks are likely assigned to the cloud that can execute them in the shortest time. Task F and G preempt task C and B, respectively. The total execution time is only 21 time unit,



a

time	0~2	3~7	9~15	15~19	19~26	26~28
Cloud 1	A	F	J	I	D	J
time		2~7	7~15	15~18	18~27	28~31
Cloud 2		B	G	B	E	K
time		2~10		15~21		28~32
Cloud 3		C		H		L

b

time	0~2				9~11			18~20
Cloud 1	A				H			L
time		2~3	3~5	5~7	9~14		14~18	18~21
Cloud 2		C	F	C	I		J	K
time		2~5		5~9	9~10	10~15	15~17	
Cloud 3		B		G	B	D	E	

Fig. 4. (a) The execution orders of three clouds with the round-robin schedule, (b) the execution orders of three clouds with the list-schedule.

which is 34% faster than the round-robin schedule. And the average execution time is 13.33, 41% faster than the round-robin schedule.

In this motivational example, we show the significant improvement by simply using CLS algorithm, even without considering the dynamic adapting scheduling. We will present the details of our algorithms in the following section.

5. Resource allocation and task scheduling algorithm

Since the manager servers neither know when applications arrive, nor whether other manager servers receive applications, it is a dynamic scheduling problem. We propose two algorithms for the task scheduling: *dynamic cloud list scheduling* (DCLS) and *dynamic cloud min–min scheduling* (AMMS).

5.1. Static resource allocation

When a manager server receives an application submission, it will first partition this application into tasks in the form of a DAG. Then a static resource allocation is generated offline. We proposed two greedy algorithms to generate the static allocation: the cloud list scheduling and the cloud min–min scheduling.

5.1.1. Cloud list scheduling (CLS)

Our proposed CLS is similar to CPNT [16]. Some definitions used in listing the task are provided as follow. The *earliest start time* (EST) and the *latest start time* (LST) of a task are shown as in Eqs. (2) and (3). The entry-tasks have EST equals to 0. And The LST of exit-tasks equal to their EST.

$$EST(v_i) = \max_{v_m \in pred(v_i)} \{EST(v_m) + AT(v_m)\} \tag{2}$$

$$LST(v_i) = \min_{v_m \in succ\, v_i} \{LST(v_m)\} - AT(v_i). \tag{3}$$

Because the cloud system concerned in this paper is heterogeneous, the execution times of a task on VMs of different clouds are not the same. $AT(v_i)$ is the average execution time of task v_i. The critical node (CN) is a set of vertices in the DAG of which EST and LST are equal. Algorithm 1 shows a function forming a task list based on the priorities.

Once the list of tasks is formed, we can allocate resources to tasks in the order of this list. The task on the top of this list will be assigned to the cloud that can finish it at the earliest time. Note that the task being assigned at this moment will start execution only when all its predecessor tasks are finished and the cloud resources allocated to it are available. After assigned, this task is removed from the list. The procedure repeats until the list is empty. An static resource allocation is obtained after this assigning procedure that is shown in Algorithm 2.

Algorithm 1 Forming a task list based on the priorities

```
Require: A DAG, Average execution time $AT$ of every task in the DAG.
Ensure: A list of tasks $P$ based on priorities.
1:  The EST of every tasks is calculated.
2:  The LST of every tasks is calculated.
3:  Empty list $P$ and stack $S$, and pull all tasks in the list of task $U$.
4:  Push the CN task into stack $S$ in the decreasing order of their LST
5:  while the stack $S$ is not empty do
6:    if $top(S)$ has un-stacked immediate predecessors then
7:      $S$ $\leftarrow$the immediate predecessor with least LST
8:    else
9:      $P \leftarrow top(S)$
10:     pop $top(S)$
11:   end if
12: end while
```

Algorithm 2 The assigning procedure of CLS

```
Require: A priority-based list of tasks $P$, $m$ different clouds, $ETM$ matrix
Ensure: A static resource allocation generated by CLS
1: while The list $P$ is not empty do
2:   $T = top(P)$
3:   Pull resource status information from all other manager servers
4:   Get the earliest resource available time for $T$, with the consideration of the dataset transferring time response from all other manager servers
5:   Find the cloud $C_{\min}$ giving the earliest estimated finish time of T, assuming no other task preempts T
6:   Assign task T to cloud $C_{\min}$
7:   Remove $T$ from $P$
8: end while
```

5.1.2. Cloud min–min scheduling (CMMS)

Min–min is another popular greedy algorithm [18]. The original min–min algorithm does not consider the dependences among tasks. So in the dynamic min–min algorithm used in this paper, we need to update the mappable task set in every scheduling step to maintain the task dependences. Tasks in the mappable task set are the tasks whose predecessor tasks are all assigned. Algorithm 3 shows the pseudo codes of the DMMS algorithm.

5.1.3. Energy-aware local mapping

A manager server uses a slot table to record execution schedules of all resources, i.e., servers, in its cloud. When an AR task is assigned to a cloud, the manager server of this cloud will first check the resource availability in this cloud. Since AR tasks can preempt best-effort tasks, the only case where an AR task is rejected is that most of the resources are reserved by some other AR tasks at the required time, no enough resources left for this task. If the AR task is not rejected, which means there are enough resources for this task, a set of servers will be reserved by this task, using the algorithm shown in Algorithm 4. The time slots for transferring the disk image of the AR task and the task execution are reserved in the slot tables of those servers. The time slots for storing and reloading the disk image of the preempted task are also reserved if preemption happens.

When a best-effort task arrives, the manager server will put it in the execution queue. Every time when there are enough VMs for the task on the top of the queue, a set of servers are selected by the algorithm shown in Alg. 5. And the manager server also updates the time slot table of those servers.

Algorithm 3 Cloud min–min scheduling (CMMS)

```
Require: A set of tasks, $m$ different clouds, $ETM$ matrix.
Ensure: A schedule generated by CMMS.
1: Form a mappable task set $P$.
2: while there are tasks not assigned do
3:    Update mappable task set $P$.
4:    for $i$: task $v_i \in P$ do
5:       Pull resource status information from all other manager servers.
6:       Get the earliest resource available time, with the consideration of the dataset transferring time response from all other manager servers.
7:       Find the cloud $C_{\min}(v_i)$ giving the earliest finish time of $v_i$, assuming no other task preempts $v_i$
8:    end for
9:    Find the task-cloud pair$(v_k, C_{\min}(v_k))$ with the earliest finish time in the pairs generated in for-loop.
10:   Assign task $v_k$ to cloud $D_{\min}(v_k)$.
11:   Remove $v_k$ from $P$.
12:   Update the mappable task set $P$
13: end while
```

Algorithm 4 Energy-aware local mapping for AR tasks

```
Require: A set of AR tasks $T$, which require to start at the same time. A set of servers $S$.
Ensure: A local mapping
1: for $t_i \in T$ do
2:    Calculate $wl_m(t_i)$
3:    if $wl(t_i) - wl_m(t_i) < \sum_{s_i \in idle}(C(s_i))$ then
4:       Schedule $wl(t_i) - wl_m(t_i)$ to the idle servers
5:    else
6:       First schedule a part of $wl(t_i) - wl_m(t_i)$ to the idle servers
7:       Schedule the rest of $wl(t_i) - wl_m(t_i)$ to the active servers, preempting the best-effort tasks
8:    end if
9: end for
10: Sort tasks in $T$ in the descending order of marginal workload, form list $L_d$.
11: Sort tasks in $T$ in the ascending order of marginal workload, form list $L_a$
12: while $T$ is not empty do
13:    $t_a = \text{top}(L_d)$
14:    if there exists a server $j$: $C(j) = wl_m(t_a)$ then
15:       Schedule the $wl_m(t_a)$ to server $j$
16:    end if
17:    $s_a = \max_{s_i \in S}(C(s_i))$.
18:    Schedule $t_a$ to $s_a$, delete $t_a$ from $T$, $L_d$, and $L_a$
19:    for $k$: $t_k \in L_a$ do
20:       if $C(s_a) > 0$ and $C(s_a) \geq wl_m(t_k)$ then
21:          Schedule $t_k$ to $s_a$, delete $t_k$ from $T$, $L_d$, and $L_a$
22:       else
23:          Break
24:       end if
25:    end for
26: end while
```

The objectives of Algorithms 4 and 5 are to minimize the number of active servers as well as the total energy consumption of the cloud. When every active server is fully utilized, the required number of active servers is minimized. When task t_i is assigned to cloud j, we define the marginal workload of this task as:

$$wl_m(t_i) = wl(t_i) \bmod C(S_j) \tag{4}$$

where S_j represents the kind server in cloud j, and $C(S_j)$ is the workload capacity of server S_j. To find the optimal local mapping,

Algorithm 5 Energy-aware local mapping for best-effort task

```
Require: A set of best-effort tasks $T$, which can start at the same time. A set of servers $S$
Ensure: A local mapping
1: for $t_i \in T$ do
2:    Calculate $wl_m(t_i)$.
3:    Schedule $wl(t_i) - wl_m(t_i)$ to the idle servers.
4: end for
5: Form a set of active servers $S_g$ that $C(s_i) > 0, \forall s_i \in S_g$.
6: Sort tasks in $T$ in the descending order of marginal workload, form list $L_d$
7: Sort tasks in $T$ in the ascending order of marginal workload, form list $L_a$
8: while $T$ is not empty do
9:    $t_a$ = top($L_d$)
10:   if there exists a server $j$ in $S_g$: $C(j) = wl_m(t_a)$ then
11:      Schedule the $wl_m(t_a)$ to server $j$
12:   end if
13:   $s_a = \max_{s_i \in S_g}(C(s_i))$
14:   if $C(s_a) < wl_m(t_a)$ then
15:      $s_a = anyidleserver$
16:   end if
17:   Schedule $t_a$ to $s_a$, delete $t_a$ from $T$, $L_d$, and $L_a$
18:   for $k$: $t_k \in L_a$ do
19:      if $C(s_a) > 0$ and $C(s_a) \geq wl_m(t_k)$ then
20:         Schedule $t_k$ to $s_a$, delete $t_k$ from $T$, $L_d$, and $L_a$
21:      else
22:         Break
23:      end if
24:   end for
25: end while
```

we group all the tasks that can be executed simultaneously, and sort them in the descending order of their marginal workloads. For each of the large marginal workload task, we try to find some small marginal workload tasks to fill the gap and schedule them on a server.

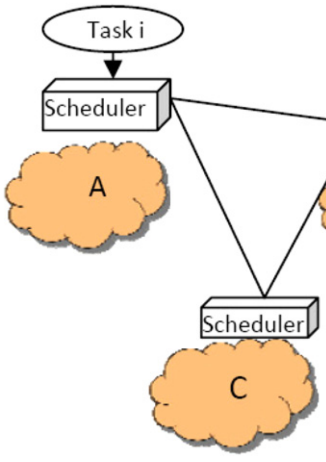
5.1.4. Feedback information

In the two static scheduling algorithms presented above, the objective function when making decision about assigning a certain task is the earliest estimated finish time of this task. The estimated finish time of task i running on cloud j, $\tau_{i,j}$, is as below:

$$\tau_{i,j} = ERAT_{i,j} + S_I/b + ETM_{i,j}. \tag{5}$$

S_I is the size of this disk image, b is the network bandwidth. $ERAT_{i,j}$ is the earliest resource available time based the information from the pull operation. It is also based on the current task queue of cloud j and the schedule of execution order. But the estimated finish time from (5) may not be accurate. For example, as shown in Fig. 5(a), we assume there are three clouds in the system. The manager server of cloud A needs to assign a best-effort task i to a cloud. According to Eq. (5), cloud C has the smallest τ. So manager server A transfers task i to cloud C. Then manager server of cloud B needs to assign an AR task j to a cloud. Task j needs to reserve the resource at 8. Cloud C has the smallest τ again. manager server B transfers task j to cloud C. Since task j needs to start when i is not done, task j preempts task i at time 8, as shown in Fig. 6. In this case, the actual finish time of task i is not the same as expected.

In order to reduce the impacts of this kind of delays, we use a feedback factor in computing the estimated finish time. As discussed previously in this paper, we assume once a task is done, the cloud will push the resource status information to the original cloud. Again, using our example in Fig. 5, when task i is done at time T_{act_fin}(=14), manager server C informs manager server A that task a



b

Task i

cloud	ERAT	SI/b	ETMi
A	2	1	10
B	3	1	8
C	4	1	6

Task j

cloud	ERAT	SI/b	ETMj
A	8	1	4
B	8	1	3
C	8	1	2

Fig. 5. Example of resource contention. (a) Two tasks are submitted to a heterogeneous clouds system. (b) The earliest resource available times (ERAT), the image transferring time (SI/b), and the execution time (EMT) of two tasks on different clouds.

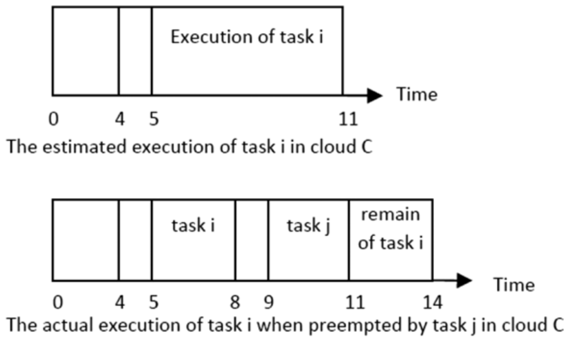

Execution of task i
Time
0
4
5
11
The estimated execution of task i in cloud C
task i
task j
remain of task i
Time
0
4
5
8
9
11
14
The actual execution of task i when preempted by task j in cloud C

Fig. 6. The estimated and the actual execution order of the cloud C.

i is done. With this information, the manager server A can compute the actual execution time $\Delta\tau_{i,j}$ of task i on cloud j:

$$\Delta\tau_{i,j} = T_{act\_fin} - ERAT_{i,j}. \tag{6}$$

And the feedback factor fd_j of cloud j is :

$$fd_j = \alpha \times \frac{\Delta\tau_{i,j} - S_I/b - ETM_{i,j}}{S_I/b + ETM_{i,j}} \tag{7}$$

α is a constant between 0 and 1. So a feedback estimated earliest finish time $\tau_{fdi,j}$ of task i running on cloud j is as follows:

$$\tau_{fd_{i,j}} = ERAT_{i,j} + (1 + fd_j) \times (S_I/b + ETM_{i,j}). \tag{8}$$

In our proposed dynamic cloud list scheduling (DCLS) and dynamic cloud min–min scheduling (DCMMS), every manager server stores feedback factors of all clouds. Once a manager server is informed that a task originally from it is done, it will update the value of the feedback factor of the task-executing cloud. For instance, in the previous example, when cloud C finishes task i and informs that to the manager server of cloud A, this manager server will update its copy of feedback factor of cloud C. When the next task k is considered for assignment, the $\tau_{fd_{k,C}}$ is computed with the new feedback factor and used as objective function.

Table 1
The mapping of job traces to applications.

Parameter in our model	Values in job traces
Task id	Job ID
Application arrival time	Min(job start time)
Task execution time	Job end time–job start time
# of CPU required by a task	Length(node list) * cpu per node

6. Experimental results

6.1. Experiment setup

We evaluate the performance of our dynamic algorithms through our own written simulation environment that acts like the IaaS cloud system. We stimulates workloads with job traces from the Parallel Workloads Archive [28]. We select three different job traces: LLNL-Thunder, LLNL-Atlas, and LLNL-uBGL. For each job tracer, we extract four values: the job ID, the job start time, the job end time, and the node list. However, job traces from the Parallel Workloads Archive do not include information about data dependencies. To simulate data dependencies, we first sort jobs by their start time. Then we group up to 64 adjacent jobs as one application, represented by a randomly generated DAG. Table 1 shows how we translate those values from job traces to the parameter we use in our application model. Note that we map the earliest job start time in an application as the arrival time of this application, since there is no record about job arrival time in these job traces. There are three data center in our simulation: (1) 1024 node cluster, with 4 Intel IA-64 1.4 GHz Itanium processors, 8 GB memory, and 185 GB disk space per node; (2) 1152 node cluster, with 8 AMD Opteron 2.4 GHz processors, 16 GB memory, and 185 GB disk space per node; (3) 2048 processors BlueGene/L system with 512 MB memory, 80 GB memory. We select these three data center configuration based on the clusters where LLNL-Thunder, LLNL-Atlas, and LLNL-uBGL job traces were obtained. Based on the information in [24], we compare the computational power of these three data center in Table 2. With the normalized performance per core, we can get the execution time of all tasks on three different

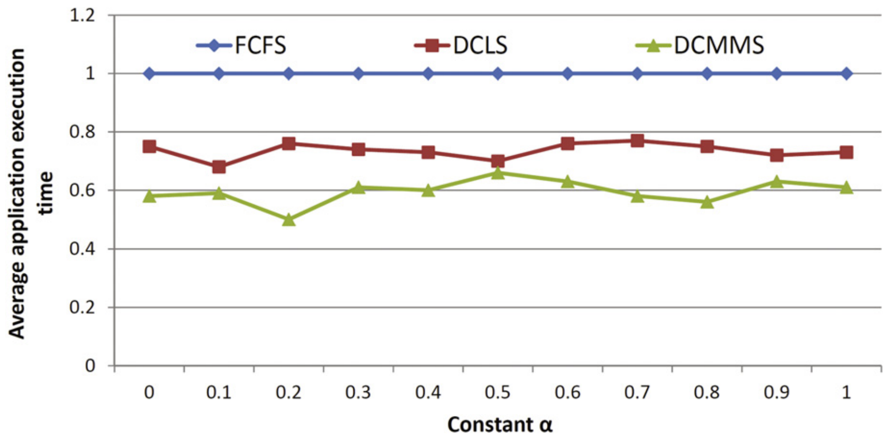


Fig. 7. Average application execution time in the loose situation.

data centers. Among these applications, 20% applications are in the AR modes, while the rest are in the best-effort modes. We assume the bandwidth between two data centers are 1 Gbps [20], the bandwidth of nodes inside the data center are 4 GBps [24], and the size of every dataset is 1 TB [27]. We run these three jobs trace separately in our simulation.

We set the arrival of applications in two different ways. In the first way, we use the earliest start time of an application in the original job trace as the arrival time of this application. We also set the required start time of an AR application as a random start time no later than 30 min after it arrives. In most of the cast, applications do not need to contend resources in this setting. We call this a *loose situation*. In the other way, we set the arrival time of applications close to each other. In this setting, we reduce the arrival time gap between two adjacent application by 100 time. It means that applications usually need to wait for resources in cloud. We call this a *tight situation*. In both these two setting, we tunes the constant α to show how the dynamic procedure impacts the average application execution time. We define the execution time of an application as the time elapses from the application is submitted to the application is finished.

6.2. Result

Fig. 7 shows the average application execution time in the loose situation. We compare our two dynamic algorithms with the First-Come-First-Serve (FCFS) algorithm [34]. We find out that the DCMMS algorithm has the shorter average execution time. And the dynamic procedure with updated information does not impact the application execution time significantly. The reason the dynamic procedure do not has a significant impact on the application execution time is that the resource contention is not significant in the loose situation. Most of the resource contentions occurs when an AR application preempts a best-effort application. So the estimated finish time of an application is usually close to the actual finish time, which limits the effect of the dynamic procedure. And the manager server does not call the dynamic procedure in most of the cases.

Fig. 8 shows that DCMMS still outperforms DCLS and FCFS. And the dynamic procedure with updated information works more significantly in the tight situation than it does in the loose situation. Because the resource contentions are fiercer in tight situation, the actual finish time of a task is often later than estimated finish time. And the best-effort task is more likely preempted some AR tasks. The dynamic procedure can avoid tasks gathering in some fast clouds. We believe that the dynamic procedure works even better in a homogeneous cloud system, in which every task runs faster in some kinds of VMs than in some other kinds.

Table 2
Comparison of three data center. The job trace LLNL-uBGL was obtained from a small uBGL, which has the same single core performance as the one shown in this table.

Data center	Peak performance (TFLOP/s)	Number of CPUs	Normalized performance per core
Thunder	23	4 096	1
Altas	44.2	9 216	0.85
uBGL(big)	229.4	81 920	0.50

Table 3
Feedback improvements in different cases.

Arrival gap reduce times	DLS	FDLS ($\alpha = 1$)	Feedback improv. (%)	DMMS	FDMMS ($\alpha = 1$)	Feedback improv. (%)
1	237.82	253.59	−6.63	206.31	223.47	−8.32
20	309.35	286.55	7.37	262.66	255.44	2.75
40	445.74	397.15	10.9	385.48	336.52	12.7
60	525.32	420.83	19.89	448.04	343.60	23.31
80	729.56	537.28	26.36	648.37	440.05	32.13
100	981.41	680.22	30.69	844.33	504.66	40.23

Table 4
Average application execution time with various percentages of AR applications in the loose situation ($\alpha = 0.8$).

	0%	20%	50%	80%	100%
FCFS	1	1	1	1	1
DCLS	0.81	0.75	0.61	0.55	0.49
DCMMS	0.77	0.56	0.52	0.46	0.44

In order to find out the relationship between resource contention and feedback improvement, we increase the resource contention by reducing the arrival time gap between two adjacent applications. We reduce this arrival time gap by 20, 40, 60, 80, and 100 times, respectively. In the setting with original arrival time gap, an application usually come after the former application is done. Resource contention is light. And when arrival time gaps are reduced by 100 times, it means during the execution of an application, there may be multiple new applications arriving. Resource contention is heavy in this case. As shown in Table 3, the improvement caused by feedback procedure increases as the resource contention become heavier.

We also test our proposed algorithms in setups with various percentages of AR applications, as shown in Tables 4 and 5. The values in the first row represent how many applications are set as the AR applications. The values in the second, the third, and the fourth row are the average application execution time, normalized by the corresponding execution time with the FCFS algorithm. From these two tables, we can observe that higher percentage of

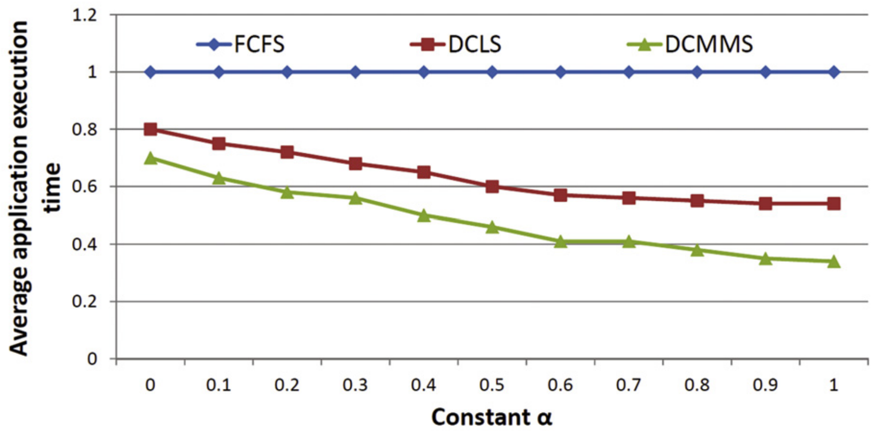


Fig. 8. Average application execution time in the tight situation.

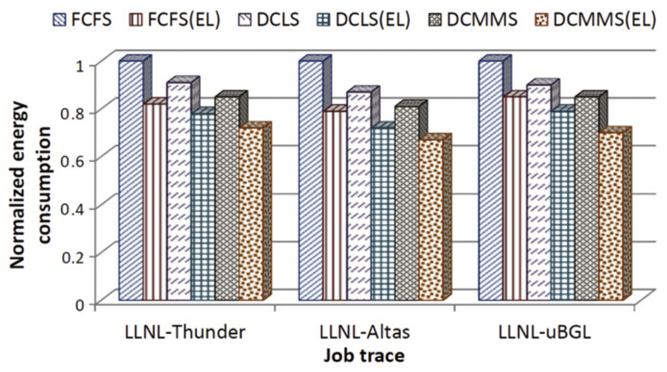


Fig. 9. Energy consumption in the loose situation. Columns without "(EL)" are schedules without energy-aware local mapping. And columns with "(EL)" are schedules with energy-aware local mapping.

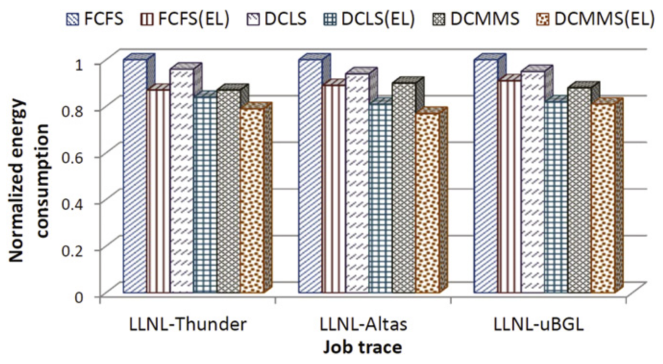


Fig. 10. Energy consumption in the tight situation. Columns without "(EL)" are schedules without energy-aware local mapping. And columns with "(EL)" are schedules with energy-aware local mapping.

Table 5
Average application execution time with various percentages of AR applications in the tight situation ($\alpha = 0.8$).

	0%	20%	50%	80%	100%
FCFS	1	1	1	1	1
DCLS	0.63	0.55	0.49	0.43	0.38
DCMMS	0.51	0.38	0.32	0.30	0.27

AR applications leads to a better improvement of the DLS and the DCMMS algorithm, compared to the FCFS algorithm, in both the loose situation and the tight situation. The reason is that more AR applications cause longer delays of the best-effort applications. By using the feedback information, our DLS and DCMMS can reduce workload unbalance, which is the major drawback of the FCFS algorithm.

Furthermore, we compare the energy consumption of three algorithms, shown in Figs. 9 and 10. Both DCLS and DCMMS can reduce energy consumption compared to the FCFS algorithm. In addition, our energy-aware local mapping further reduce the energy consumption significantly, in all three algorithms.

In the future work, we will evaluate our proposed mechanism in existing simulators, so that results can be reproduced easier by other researchers. In addition, we will investigate the implementation of our design in the real-world cloud computing platform. A reasonable way to achieve this goal is to combine our design with the Hadoop platform [15]. The multi-cloud scheduling mechanism and algorithms in our design can be used on the top of the Hadoop platform, distributing applications in the federated multi-cloud platform. When a give task is assigned to a cloud, the Hadoop will be used to distribute tasks to multiple nodes. And our proposed energy-aware local mapping design can be implemented in the Hadoop Distributed File System, which enables the "rack awareness" feature for data locality inside the data center.

7. Conclusion

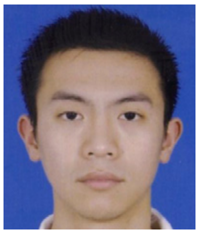
The cloud computing is emerging with rapidly growing customer demands. In case of significant client demands, it may be necessary to share workloads among multiple data centers, or even multiple cloud providers. The workload sharing is able to enlarge the resource pool and provide even more flexible and cheaper resources. In this paper, we present a resource optimization mechanism for preemptable applications in federated heterogeneous cloud systems. We also propose two novel online dynamic scheduling algorithms, DCLS and DCMMS, for this resource allocation mechanism. Experimental results show that the DCMMS outperforms DCLS and FCFS. And the dynamic procedure with updated information provides significant improvement in the fierce resource contention situation. The energy-aware local mapping in our dynamic scheduling algorithms can significantly reduce the energy consumptions in the federated cloud system.

Acknowledgments

This work was supported in part by the University of Kentucky Start Up Fund and NSFC 61071061; NSFC 61170077; SZ-HK Innovation Circle proj. ZYB200907060012A, NSF GD:10351806001000000, S & T proj. of SZ JC200903120046A; the NSF CNS-0969013, CNS-0917021, and CNS-1018108; the NSF CNS-0915762 (CSR), CCF-08452578 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), Auburn Start Up grant and Intel gift (2005-04-070); NSFC 61070002 and MOE-INTEL-10-02.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, et al., Above the clouds: a Berkeley view of cloud computing, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.
[2] M. Aron, P. Druschel, W. Zwaenepoel, Cluster reserves: a mechanism for resource management in cluster-based network servers, in: Proceedings of the ACM Sigmetrics, Santa Clara, California, USA, 2000.
[3] A.I. Avetisyan, R. Campbell, M.T. Gupta, I. Heath, S.Y. Ko, G.R. Ganger, et al., Open Cirrus a global cloud computing testbed, IEEE Computer 43 (4) (2010) 35–43.
[4] D. Cearley, G. Phifer, Case studies in cloud computing, http://www.gartner.com/it/content/1286700/1286717.
[5] J.S. Chase, D.E. Irwin, L.E. Grit, J.D. Moore, S. Sprenkle, Dynamic virtual clusters in a grid site manager, in: International Symposium on High-Performance Distributed Computing, HPDC, Seattle, Washington, USA, 2003, pp. 90–103.
[6] H.J.M.Y. Chi, H. Hacigumus, Performance evaluation of scheduling algorithms for database services with soft and hard SLAs, in: International Workshop on Data Intensive Computing in the Clouds, 2011, pp. 1–10.
[7] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM 1 (2008) 107–113.
[8] A. Dogan, F. Ozguner, Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 308–323.
[9] W. Emeneker, D. Stanzione, Efficient virtual machine caching in dynamic virtual clusters, in: SRMPDS Workshop of International Conference on Parallel and Distributed Systems, Hsinchu, Taiwan, 2007.
[10] N. Fallenbeck, H. Picht, M. Smith, B. Freisleben, Xen and the art of cluster scheduling, in: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, Tampa, Florida, USA, 2006, p. 4.
[11] T. Forell, D. Milojicic, V. Talwar, Cloud management challenges and opportunities, in: IEEE International Symposium on Parallel and Distributed, 2011, pp. 881–889.
[12] I.T. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, A. Roy, A distributed resource management architecture that supports advance reservations and co-allocation, in: International Workshop on Quality of Service, IWQoS, London, UK, 1999, pp. 27–36.
[13] G. Garzoglio, T. Levshina, P. Mhashilkar, S. Timm, ReSS: a resource selection service for the open science grid, in: International Symposium on Grid Computing, 2007, pp. 1–10.
[14] C. Germain-Renaud, O. Rana, The convergence of clouds, grids, and autonomics, IEEE Internet Computing 13 (6) (2009) 9.
[15] Apache Hadoop, http://wiki.apache.org/hadoop/.
[16] T. Hagras, J. Janecek, A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems, Parallel Computing 31 (7) (2005) 653–670.
[17] H. Huang, L. Wang, P&p: a combined push–pull model for resource monitoring in cloud computing environment, in: IEEE International Conference on Cloud Computing, Miami, Florida, USA, 2010, pp. 260–266.
[18] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, Journal of the ACM 24 (2) (1977) 280–289.
[19] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, K.G. Yocum, Sharing networked resources with brokered leases, in: USENIX Annual Technical Conference, Boston, Massachusetts, USA, 2006.
[20] W. Jiang, Y. Turner, J. Tourrilhes, M. Schlansker, Controlling traffic ensembles in open cirrus, https://www.hpl.hp.com/techreports/2011/HPL-2011-129.pdf.
[21] K. Keahey, T. Freeman, Contextualization: providing one-click virtual clusters, in: IEEE International Conference on eScience, Indianapolis, Indiana, USA, 2008.
[22] M.A. Kozuch, M.P. Ryan, R. Gass, S.W. Schlosser, D. O'Hallaron, et al. Cloud management challenges and opportunities, in: Workshop on Automated control for datacenters and cloud, 2009, pp. 43–48.
[23] H. Liu, D. Orban, GridBatch: cloud computing for large-scale data-intensive batch applications, in: IEEE International Symposium on Cluster Computing and the Grid, 2008, pp. 295–305.
[24] Open computing facility - ocf, https://computing.llnl.gov/?set=resources&page=OCF-resource.
[25] R. Moren-Vozmediano, R.S. Montero, I.M. Llorente, Elastic management of cluster-based services in the cloud, in: Workshop on Automated control for datacenters and cloud, 2009, pp. 19–24.
[26] C. Moretti, J. Bulosan, P.J. Thain, D. Flynn, All-pairs: an abstraction for data-intensive cloud computing, in: IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–11.
[27] A. Pavlo, E. Paulson, A. Rasin, D.J. Ababi, D. DeWitt, S. Madden, M. Stonebraker, A comparison of approaches to large-scale data analysis, in: SIGMOD, 2009, pp. 1–14.
[28] Parallel workloads archive, www.cs.huji.ac.il/labs/parallel/workload/.
[29] M. Qiu, M. Guo, M. Liu, C.J. Xue, L.T. Yang, E.H.-M. Sha, Loop scheduling and bank type assignment for heterogeneous multi-bank memory, Journal of Parallel and Distributed Computing (JPDC) 69 (6) (2009) 546–558.
[30] M. Qiu, E.H.-M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Transactions on Design Automation of Electronic Systems (TODAES) 14 (2) (2009) 1–30.
[31] B.P. Rimal, E. Choi, I. Lumb, A taxonomy and survey of cloud computing systems, in: International Joint Conference on INC, IMS and IDC, 2009, pp. 44–51.
[32] P. Ruth, P. McGachey, D. Xu, Viocluster: virtualization for dynamic computational domains, in: Proceedings of the IEEE International Conference on Cluster Computing, Boston, Massachusetts, USA, 2005, pp. 1–10.
[33] P. Ruth, J. Rhee, D. Xu, R. Kennell, S. Goasguen, Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure, in: IEEE International Conference on Autonomic Computing, 2006, pp. 5–14.
[34] W. Smith, I. Foster, V. Taylor, Scheduling with advanced reservations, in: IEEE International Parallel and Distributed Processing Symposium, Cancun, Mexico, 2000, pp. 127–132.
[35] B. Sotomayor, K. Keahey, I. Foster, Combining batch execution and leasing using virtual machines, in: Proceedings of the 17th International Symposium on High Performance Distributed Computing, Boston, Massachussets, USA, 2008, pp. 87–96.
[36] B. Sotomayor, R. Llorente, I. Foster, Resource leasing and the art of suspending virtual machines, in: 11th IEEE International Conference on High Performance Computing and Communications, Seoul, Korea, 2009, pp. 59–68.
[37] B. Sotomayor, R. Montero, I. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing 13 (5) (2009) 14–22.
[38] Amazon AWS, http://aws.amazon.com/.
[39] GoGrid, http://www.gogrid.com/.
[40] RackSpace, http://www.rackspacecloud.com/.
[41] Microsoft cloud, http://www.microsoft.com/en-us/cloud/.
[42] IBM cloud, http://www.ibm.com/ibm/cloud/.
[43] Google apps, http://www.google.com/apps/intl/en/business/index.html.
[44] HP cloud, http://www8.hp.com/us/en/solutions/solutionsdetail.html?compURI=tcm:245-300983\&pageTitle=cloud.
[45] Eucalyptus, http://www.eucalyptus.com/.
[46] RESERVOIR, www.reservoir-fp7.eu.
[47] E. Walker, J. Gardner, V. Litvin, E. Turner, Dynamic virtual clusters in a grid site manager, in: Proceedings of Challenges of Large Applications in Distributed Environments, Paris, France, 2006, pp. 95–103.
[48] X. Wang, J. Zhang, H. Liao, L. Zha, Dynamic split model of resource utilization in mapreduce, in: International Workshop on Data Intensive Computing in the Clouds, 2011, pp. 1–10.
[49] M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, I. Foster, Swift: a language for distributed parallel scripting, Parallel Computing 37 (9) (2011) 633–652.
[50] T. Wood, A. Gerber, K. Ramakrishnan, J. van der Merwe, The case for enterprise-ready virtual private clouds, in: Workshop on Hot Topics in Cloud Computing, San Diego, Califoria, USA, 2009.
[51] S. Zanikolas, R. Sakellariou, A taxonomy of grid monitoring systems, Future Generation Computer systems 1 (2005) 163–188.



Jiayin Li received the B.E. and M.E. degrees from Huazhong University of Science and Technology (HUST), China, in 2002 and 2006, respectively. And now he is pursuing his Ph.D. degree in the Department of Electrical and Computer Engineering (ECE), University of Kentucky. His research interests include software/hardware co-design for embedded system and high performance computing.

Meikang Qiu received the B.E. and M.E. degrees from Shanghai Jiao Tong University, China. He received the M.S. and Ph.D. degrees in Computer Science from University of Texas at Dallas in 2003 and 2007, respectively. He had worked at Chinese Helicopter R&D Institute and IBM. Currently, he is an assistant professor of ECE at University of Kentucky. He is an IEEE Senior member and has published 140 journal and conference papers, including 15 IEEE/ACM Transactions. He is the recipient of the ACM Transactions on Design Automation of Electronic Systems (TODAES) 2011 Best Paper Award. He also received three other best paper awards (IEEE EUC'09, IEEE/ACM GreenCom'10, and IEEE CSE'10) and one best paper nomination. He also holds 2 patents and has published 3 books. He has also been awarded SFFP Air Force summer faculty in 2009. He has been on various chairs and TPC members for many international conferences. He served as the Program Chair of IEEE EmbeddCom'09 and EM-Com'09. His research interests include embedded systems, computer security, and wireless sensor networks.

Zhong Ming is a professor at College of Computer and Software Engineering of Shenzhen University. He is a member of a council and senior member of China Computer Federation. His major research interests are software engineering and embedded systems. He led two projects of National Natural Science Foundation, and two projects of Natural Science Foundation of Guangdong province, China.

Gang Quan is currently an Associate Professor with the Electrical and Computer Engineering Department, Florida International University, Miami. He received the B.S. degree from the Tsinghua University, Beijing, China, the M.S. degree from the Chinese Academy of Sciences, Beijing, and the Ph.D. degree from the University of Notre Dame, Notre Dame, IN. His research interests include real-time system, power/thermal aware design, embedded system design, advanced computer architecture and reconfigurable computing. Prof. Quan received the NSF CAREER award in 2006.

Xiao Qin received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska-Lincoln in 2004. He is currently an associate professor of computer science at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He won an NSF CAREER award in 2009. His research interests include parallel and distributed systems, real-time computing, storage systems, fault tolerance, and performance evaluation. His research is supported by the U.S. National Science Foundation, Auburn University, and Intel Corporation. He is a senior member of the IEEE and the IEEE Computer Society.

Zonghua Gu received his Ph.D. degree in computer science and engineering from the University of Michigan at Ann Arbor in 2004. He is currently an Associate Professor in the Zhejiang University, China. His research interests include real-time embedded systems.