

Feedback Dynamic Algorithms for Preemptable Job Scheduling in Cloud Systems

Jiayin Li¹ Meikang Qiu¹ Jianwei Niu² Wenzhong Gao³ Ziliang Zong⁴ Xiao Qin⁵

¹ Dept. of Elec. and Comp. Engr., University of Kentucky, Lexington, KY 40506, USA.

² School of Computer Science and Engineering, Beihang University, Beijing 100191, China.

³ Dept. of Elec. and Comp. Engr., University of Denver, Denver, CO 80210, USA

⁴ Dept. of Math. and Comp. Sci., South Dakota School of Mines, Rapid City, SD 57702, USA

⁵ Dept. of Comp. Sci. and Software Engr., Auburn University, Auburn, AL 36849, USA

Abstract—An infrastructure-as-a-service cloud system provides computational capacities to remote users. Parallel processing in the cloud system can shorten the execution of jobs. Parallel processing requires a mechanism to scheduling the executions order as well as resource allocation. Furthermore, a preemptable scheduling mechanism can improve the utilization of resources in clouds. In this paper, we present a preemptable job scheduling mechanism in cloud system. We propose two feedback dynamic scheduling algorithms for this scheduling mechanism. We compare these two scheduling algorithms in simulations. The results show that the feedback procedure in our algorithms works well in the situation where resource contentions are fierce.

Keywords: Cloud computing, dynamic scheduling, feedback, preemptable scheduling

I. INTRODUCTION

In infrastructure-as-a-service (IaaS) cloud computing, computational resources are leased to remote users who can get the control of those resources [1]. As the free and efficient virtualization solutions, such as the Xen hypervisor (www.xen.org), became available, turning over control to remote users is no longer fraught with danger [2]. IaaS provides a large amount of computational capacities to users in a flexible and efficient way. For example, Amazon's Elastic Compute Cloud (EC2: www.amazon.com/ec2/) offers virtual machines (VMs) for 0.10 US dollar per hour. An IaaS cloud enables on-demand provisioning of resources in the form of VMs deployed in the provider's data center, minimizing associated capital costs. The on-demand feature in IaaS lets consumers add or remove resources from clouds to meet peak or fluctuating service demands and pay only the capacity used. Taking advantage of the large computing capacities provided by IaaS clouds,

The email addresses of the authors are {jli6,mqiu}@engr.uky.edu, niujianwei@buaa.edu.cn, gaowenz@yahoo.com, Ziliang.Zong@sdsmt.edu, and xqin@auburn.edu.

This work was supported in part by State Key Laboratory of Software Development Environment Grant No. BUAA SKLSDE-2010ZX-13; the NSF Grants CNS-0915762 (CSR); CCF-0845257 (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 startup grant and Intel gift (Number 2005-04-070).

parallel processing can strongly improve the performance of the intensive computation applications.

When an application is submitted to the clouds, it is usually partitioned into several tasks. When applying parallel processing in executing these tasks, we need to consider the following questions: 1) how to allocate resources to the tasks; 2) in what order the clouds should execute the tasks; and 3) how to schedule overheads when VMs prepare, terminate or switch tasks. Resource allocation and task scheduling can solve these three problems. Resource allocation and task scheduling have been studied in high performance computing [3] and in embedded systems [4]. However, the autonomic feature within clouds [5] and the VM implementation require different algorithms for resource allocations and task scheduling in the IaaS cloud computing.

The two major contributions of this study are:

- We developed a task scheduling mechanism in IaaS cloud system, which enables preemptable task scheduling.
- We proposed two feedback dynamic scheduling algorithms for scheduling tasks. Their goal is to generate scheduling with the shortest average execution time of jobs.

In section II, we discuss works related to this topic. In section III, models for task scheduling in IaaS cloud computing system are presented. We propose our algorithms in section IV, followed by experimental result in section V. Finally, we give the conclusion in section VI.

II. RELATED WORK

Computational resource management in cloud computing has been recently studied in the literature. In [6], the authors proposed an image caching mechanism to reduce the overhead of loading disk image in virtual machines. The authors of [7] presented a dynamic approach to create virtual clusters to deal with the conflict between parallel and serial jobs. In this approach, job load is adjusted automatically without running time predictions. A system which can automatically scale its utilization of infrastructure resources is designed in [8]. Another resource sharing system which can trade machines in different domains without infringing autonomy of them is developed in [9]. Studies described above, however, do not

consider the issue of preemptable task scheduling. In [10], a suspend/resume mechanism is used to improve utilization of physical resource. The overhead of suspending/resume is modeled and scheduled explicitly. The VMs model considered in [10] is homogeneous, so the scheduling algorithm is not applicable in heterogeneous VMs models. The study presented in [11] focuses on scheduling in heterogeneous mobile ad hoc grid environments. However the scheduler algorithms can not be used in cloud computing.

III. MODEL AND BACKGROUND

A. Cloud systems

In this paper, we consider an infrastructure-as-a-service (IaaS) cloud system. In this kind of system, several cloud providers participate. These cloud providers deliver basic on-demand storage and computing capacities over the internet. The provision of these computational resources is in the form of virtual machines (VMs) deployed in a provider's data center. These resources form the same provider form a cloud. A virtual machine is an abstract unit of storage and computing 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. In addition, the computational characteristics of different CPU may not be the same.

In our proposed cloud scheduling mechanism, every provider has a scheduler software running in its data center. These schedulers know the current statuses of VMs in their own clouds. The schedulers communicate with each other. Therefore they can make schedule decision when using the information like the earliest resource available time in a certain cloud. Due to the security issues, we limit the types of communication functions among schedulers as following. The scheduler of cloud A can send a task checking request to the scheduler of cloud B . The scheduler of B responds with the earliest available time of all required resources, based on the current status of B 's resources. When B finishes a task which was transferred from the scheduler of A , the scheduler of B will inform the A 's scheduler that the task is finished.

When a job is submitted to a cloud, the scheduler first partitions the job into several tasks. Then for each task in this job, the scheduler decides a cloud used to execute this task based on the information from all other schedulers. If the scheduler assigns a task to its own cloud, it will store the task in a queue. When the resources and the data are ready, this task's execution begins. If the scheduler of cloud A assigns a task to cloud B , B 's scheduler 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 scheduler of B will reject the task.

Before a task in the queue of a scheduler is about to be executed, the scheduler transfer a disk image to all the computing nodes which provide enough VMs for task execution. We assume that all required disk images are stored in a data

center from which the images can be transferred to any clouds as needed. Assuming the size of this disk image is S_I , we use the multicasting and 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.

B. The resource allocation model

In cloud computing, there are three 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.
- Immediate: When a client submits a request, either the resources are provisioned immediately, or the request is rejected, based on the resource availabilities.

A lease of resource is implemented as a set of VMs. 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.

As shown in [12], combining AR and best-effort in a preemptable fashion can overcome the utilization problems. In this paper, we assume that a few of jobs submitted in the cloud system are in the AR mode, while the rest of the jobs are in the best-effort mode. The jobs in the AR mode have higher priorities, thereby being able to preempt the executions of the best-effort jobs.

When an AR task i needs to preempt a best-effort task j , the VMs have to suspend task j and restore the current disk image of task j in a specific disk space before the scheduler transfers the disk image of tasks i to the VMs. Assuming the speed of copying in the disk is S_c , and the size of j 's disk image is M_j , then the preparation overhead is M_j/S_c . When the task i finishes, the VMs will resume the execution of task j . Similar to the preparation overhead, the reloading overhead, which is caused by reloading the disk image of task j back, is also M_j/S_c . We assume that there is a specific disk space in every node for storing the disk image of suspended task.

C. Job model

In this paper, we use the *Directed Acyclic Graphs* (DAG) to model jobs. A DAG $T = (V, E)$ consists of a set of vertices V , each of which represents a task in the job, and a set of edges E , showing the dependencies among the 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 task 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-tasks, and tasks without immediate successors are exit-tasks. In order to describe the

difference between VMs' computational characteristics, we use an $M \times N$ execution time matrix (ETM) M to indicate the execution time of M types of tasks running on N types of VMs.

IV. SCHEDULING ALGORITHM

Since the schedulers neither know when jobs arrive, nor whether other schedulers receive jobs, we are addressing a dynamic scheduling problem. We propose two algorithms to solve the task scheduling problem: feedback dynamic list scheduling and feedback dynamic min-min scheduling. Once a job is submitted to a scheduler, it will be partitioned into tasks in the form of a DAG. In feedback dynamic list scheduling, a queue of tasks is generated with the consideration of the data dependencies. Tasks will be scheduled dynamically. Since one scheduler does not know the tasks executions of other clouds in details and the AR tasks may preempt the best-effort tasks, the actual finish time of a certain task may not be the same as expected. So a feedback process is used when making assigning decisions.

A. Dynamic list scheduling (DLS)

Our proposed DLS is similar to *Critical Path on a Processor* (CPOP) [3]. In this DLS algorithm, we first need to generate a priority list of tasks. This priority list represents an execution order of tasks, meaning that for a certain task i , its parent tasks should be placed in front of i , and its child tasks should be placed behind i . We used the listing method proposed in [3] to generate the priority list.

Once the list of tasks is formed, we can assign tasks to clouds in the order of this list. When the task on the top of this list is ready to run, it will be assigned to the cloud that can finish it at the earliest time. Then this task is removed from the list. The procedure repeats until the list is empty. An optimal schedule is obtained according to this assigning procedure which is shown in Algorithm 1 below.

Algorithm 1 The assigning procedure of DLS

Require: A priority-based list of tasks P , m different clouds, ETM matrix

Ensure: A schedule generated by DLS.

```

1: while The list  $P$  is not empty do
2:    $T = top(P)$ 
3:   if All the predecessor tasks of  $T$  are done then
4:     Send task checking requests of  $T$  to all other schedulers
5:     Receive the earliest resource available time responses for  $T$  from all other schedulers.
6:     Find the cloud  $C_{min}$  giving the earliest estimated finish time of  $T$ , assuming no other task preempts  $T$ 
7:     Assign task  $T$  to cloud  $C_{min}$ .
8:     Remove  $T$  from  $P$ 
9:   else
10:    Wait for the finishes of the predecessor tasks of  $T$ 
11:   end if
12: end while

```

B. Dynamic min-min scheduling (DMMS)

Min-min is another popular greedy algorithm [13]. The original min-min algorithm does not consider the dependencies among tasks. Thus in the dynamic min-min algorithm used in this paper, we need to update the mappable task set in every step to maintain the task dependencies. Tasks in the mappable task set are the tasks whose predecessor tasks are all finished. Algorithm 2 shows the pseudo codes of the DMMS algorithm.

Algorithm 2 Dynamic min-min scheduling (DMMS)

Require: A set of tasks, m different clouds, ETM matrix

Ensure: A schedule generated by DMMS.

```

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:     Send task checking requests of  $v_i$  to all other schedulers
6:     Receive the earliest resource available time responses from all other schedulers
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

```

C. Local mapping

A scheduler uses a slot table to record execution schedule of all VMs in its cloud. When an AR task is assigned to a cloud, this scheduler will first check the resource availability in the cloud. If there are enough resources for this AR task, a set of required VMs are selected arbitrarily. The time slots for disk image transferring and the task execution are reserved in the slot table accordingly. When a best-effort task arrives, the scheduler will put it in the execution queue. Every time when there are enough VMs, the task on the top of the queue will be executed. And the scheduler also updates the time slot table of those VMs.

D. Feedback procedure

Due to the cloud resource contentions and the preemptable resource allocations, the estimated finished time of a best-effort task may not be accurate. We introduce a feedback procedure in scheduling task to reduce the impact of resource contentions. When a task is finished in a certain cloud, we record the time gap between the estimated finish time and the actual finish time. And in the scheduling of a following task, we adjust the estimated time of this task running on this cloud by adding the product of this time gap and a constant α . Therefore, the fiercer resource contention the cloud has, the longer estimated finish times of tasks running on it, leading to fewer tasks actually scheduling to this cloud.

V. EXPERIMENTAL RESULTS

A. Experiment setup

We evaluate the performance of the feedback dynamic algorithms through simulations. Each simulation run (total 10 runs) has 64 unique jobs, and each application is composed of up to 16 tasks. For each task, the maximum fan-in and fan-out are both 3. We simulate 4 clouds in a cloud system, each of which has up to 100 VMs. Those 64 jobs will be submitted to random clouds at arbitrary arrival times. Among these 64 jobs, 12 jobs are in the AR modes, while the rest are in the best-effort modes. Since we focus on the scheduling algorithms, we do our simulations locally without implementing in any existing cloud system or using VM interface APIs.

We set the arrival of jobs in two different ways. In the first way, we spread the arrival time of jobs widely over time so that, in most of the cases, jobs do not need to contend resources in the cloud. We refer to this condition as a loose situation. In the other way, we set the arrival times of jobs close. It means that jobs usually need to wait for resources in the cloud. We name this situation as a tight situation. In both those two settings, we tune the constant α in computing the feedback factor to show how the feedback procedure impacts the average execution time.

B. Result

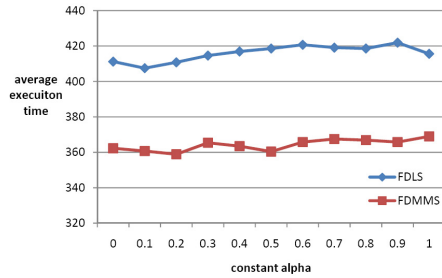


Fig. 1. average execution time in loose situations

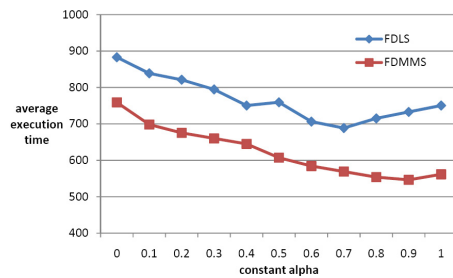


Fig. 2. average execution time in tight situations

Figure 1 shows the average execution time in the loose situation. We find out that the FDMMS algorithm has the shorter average execution time. The feedback procedure works the best when the constant α is 0.1. However, the average execution time do not change much when α changes from 0 to 1. The reason the feedback procedure do not have significant

impact on average execution time is that the start time of a certain task depends on when its predecessors are done, not the earliest resource available time in the loose situation. It means tasks usually get the resources immediately when they are ready. So in this case, the feedback factor is zero in most of time. In loose situation, most of the resource contentions occur when an AR job preempts a best-effort job.

Figure 2 shows that FDMMS still outperforms FDLS. Specifically the feedback procedure works more significantly in tight situation than it does in loose situation. Because the resource contentions are fiercer in tight situation, the actual start time of a task is often later than estimated start time. And the best-effort task is more likely preempted by some AR tasks. The feedback procedure can avoid tasks gathering in some fast clouds. We believe that the feedback procedure works even better in a homogeneous cloud system.

VI. CONCLUSION

In this paper, we present a preemptable job scheduling mechanism in IaaS cloud systems. In addition we propose two feedback dynamic scheduling algorithms for this scheduling mechanism. Simulation results show that our FDMMS works better than FDLS. And the feedback procedure works better in the situation where resource contention is fierce.

REFERENCES

- [1] R. Grossman, "The case for cloud computing," *IT Professional*, vol. 11, no. 2, pp. 23–27, March–April 2009.
- [2] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.
- [3] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," in *The 18th International Parallel and Distributed Processing Symposium*, 2003.
- [4] M. Qiu and E. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 2, pp. 1–30, 2009.
- [5] C. Germain-Renaud and O. Rana, "The Convergence of Clouds, Grids, and Autonomics," *IEEE Internet Computing*, p. 9, 2009.
- [6] W. Emenecker and D. Stanzione, "Efficient Virtual Machine Caching in Dynamic Virtual Clusters." in *SRMPDS Workshop, ICAPDS, 2007*.
- [7] N. Fallenbeck, H. Picht, M. Smith, and B. Freisleben, "Xen and the art of cluster scheduling," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006, p. 4.
- [8] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure," in *IEEE International Conference on Autonomic Computing*, 2006.
- [9] P. Ruth, P. McGachey, and D. Xu, "Viocluster: Virtualization for dynamic computational domains," in *Proceedings of the IEEE International Conference on Cluster Computing*, 2005.
- [10] B. Sotomayor, R. Llorente, and I. Foster, "Resource Leasing and the Art of Suspending Virtual Machines," in *11th IEEE International Conference on High Performance Computing and Communications*, pp. 59–68.
- [11] S. Shivle, R. Castain, H. Siegel, A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor *et al.*, "Static mapping of subtasks in a heterogeneous ad hoc grid environment," in *13th IEEE Heterogeneous Computing Workshop*, 2004.
- [12] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High performance distributed computing*, 2008, pp. 87–96.
- [13] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the ACM*, pp. 280–289, 1977.