ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 13-18 (2014)
Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand

HEFT based Cloud Auto-Scaling Algorithm with Budget Constraints

Jiping Zhou, Chunhua Gu, Feng Wan
East China University of Science and Technology, China, zhoujip@yeah.net
East China University of Science and Technology, China, chgu@ecust.edu.cn
East China University of Science and Technology, China, vann@ecust.edu.cn

Abstract : Cloud computing nowadays is playing major role in processing enterprise and personal applications. Its dynamic scalability enables users to scale up/down underlying infrastructure in response to dynamic performance behaviors. This benefit provides the basis for solving performance-resource mapping problem when deploying an application on cloud. Many researches on exploring efficient performance-resource mapping mechanisms have been launched. They generally focus on minimizing jobs' makespan with limited resources but few consider budget constraint as a parameter for their mechanisms. However, budget constraint is a critical factor that should be considered especially when application owner has limited budget. In this paper, we present a HEFT based cloud auto-scaling algorithm to minimize jobs' makespan within budget and evaluate it by comparing to other four approaches that are not or partially based on HEFT. Results have shown that proposed algorithm optimizes the application performance with limited budget compared to other four approaches and works well even if the budget is not enough to make sure all jobs finished within their deadline.

Key words : Cloud Computing, DAG, HEFT, Auto-Scaling, Budget, Job Deadline.

INTRODUCTION

With the maturity of the cloud computing technology, more and more cloud platforms have mushroomed in the market. Cloud providers charge users using pay-as-you-go model and cloud computing is being used to deliver on demand storage and processing power. With the benefits of flexibility and security, more and more application owners run their applications on cloud. Most of the applications are based on an application model that is composed of multiple decoupled independent services that can run in different virtual machines (VMs). However, mapping performance requirements to the underlying resources in the cloud automatically is challenging especially when it comes to the practical considerations like multiple VM instance types, specific performance requirement, user budget constraints and etc**.** The key cloud platforms in cloud computing field including Amazon AWS [1], Microsoft Azure [2], Google AppEngine[3] and RightScale [4] offer a variety of pre-packaged services for monitoring, managing and provisioning resources and application services. These services try to help users automatically scale VMs up/down to solve the performance-resource mapping problem. However, most of them are based on resource utilization, such as " Add 2 small instances when the average CPU is above 80% for more than 5 minutes." Those simple mechanisms are far from user expectations when the application models are complex.

A reasonable auto-scaling mechanism should be based on real cloud platform characteristics and consider the practical factors including multiple VM instance types, unique performance requirement, VM startup time, cost, job deadlines, user budget constraints, etc.

In this paper, we propose an auto-scaling mechanism based on Heterogeneous-Earliest-Finish-Time algorithm (HEFT) [5] to minimize jobs' makespan within predefined limited user budget.

RELATED WORK

Related work mainly focuses on researches of cloud auto-scaling and auto-scheduling mechanisms which deal with VM scaling up/down and scheduling interdependent tasks of a workflow application. These researches try to find out a mechanism to minimize makespan of the application job using a general application model in which an application is represented by a directed acyclic graph (DAG). Some of them go further to consider cost and deadline as a parameter for their algorithms. But few consider how to optimize application's performance with budget constraints.

For auto-scaling mechanisms, Marshall [6] proposes a model capable of monitoring the demand of applications and responding by acquiring or releasing cloud nodes to provide dynamic computing power. Assuncao and Costanzo [7] propose several instance acquisition policies and evaluate their performance under different workloads. These researches are only focused on improving performance and do not address the financial costs or user budget. A scheme is proposed in [8] to achieve VM level auto-scaling of cloud resources with cost consideration for the web application providers. However, it only applies to typical web application model without budget consideration and can't apply to a general service-oriented application model. Some researches [9][10][11][12] take a more general application model to explore auto-scaling mechanisms. Ming Mao and Marty Humphrey [11] explore auto-scaling mechanism to minimize cost and meet application deadlines in cloud workflows. It doesn't consider user budget constraint and doesn't apply to situations when budget is limited. The mechanism proposed in [12] takes into account user budget, but it is only suitable for a batch-queue application model. Haluk and Salim [5] propose the HEFT algorithm which is an application scheduling algorithm for a bounded number of

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 13-18 (2014)
Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand

heterogeneous processors. This algorithm provides the basis for the auto-scaling algorithm the paper presents. Nitish and Sarbjeet [13] explore HEFT based workflow scheduling algorithm for cost optimization in hybrid clouds. They don't consider budget constraints either and ignore some practical factors like different VM types and VM instance startup time.

CLOUD AUTO-SCALING ALGORITHM

Architecture

To support proposed algorithm in this paper, we have implemented our cloud auto-scaling mechanism in OpenStack cloud (an open source cloud platform). Figure 1 shows the architecture of our implementation. The architecture includes seven components. They are performance monitor, status table, job dispatcher, task scheduler, scale worker, back worker and VM manager. These components work together to provide functional auto-scaling mechanism to scale up/down VM instances automatically in respond to dynamic workload pattern.

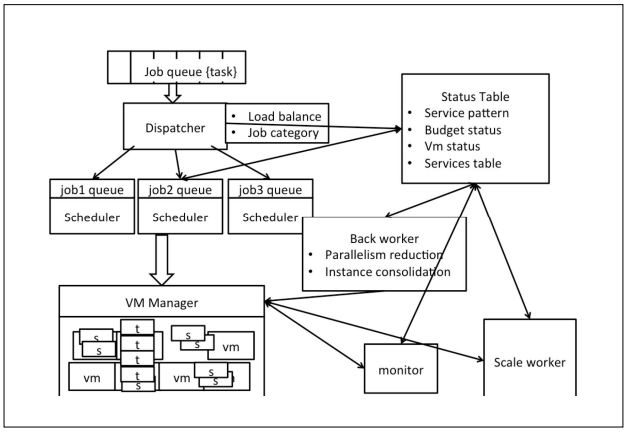

Fig 1: Architecture of Cloud auto-scaling in OpenStack

1) Performance monitor: it observers the current VM usage information and task queues status on typical services of VM instances, collects actual task processing time, updates the status table periodically. The precise dynamic status information provided by monitor is critical for making auto-scaling and auto-scheduling decisions.

2) Status table: it contains two data structures. One is static configuration file, which records DAG of application, job deadlines, budget constraint and computation cost matrix, the other records dynamic information including current instance status, task queue, workload pattern, etc. The dynamic information is updated by the performance monitor, and is used when making scheduling and scaling policies.

3) Job dispatcher: it's responsible for load balance, classifying the incoming jobs and dispatching them to the corresponding job type queues. The job type queue will receive jobs of the same type. In this way we can manage jobs of the same type more efficiently.

4) Task scheduler and scale worker: They work for making auto-scheduling and auto-scaling decisions based on the information provided by Status table. Proposed algorithm will be implemented in this layer.

5) VM manager: it provides APIs to manage the VM instances in the VM pool. The APIs can be generally grouped into two categories. One is for normal VM instance management. The other is designed to monitor the VM instances running status. VM manager hides all cloud provider details and can be easily replaced with other cloud adapters.

6) Back Worker: For tasks running on VMs, there may be a possibility that the task have to wait for other tasks for processing because of the dependency relationship in a DAG, the task could waste a significant portion of a purchased instance-hour. Back worker will find out situations like this and reduce task concurrency and consolidate VM instances to improve instance utilization rate and help users spend money on more urgent situations.

Solution

Proposed algorithm focuses on performance optimization with budget constraints. To get the highest VM performance, we have to find out a mechanism to dicide how and when to scale up/down a VM instance.

A. Problem Definition

We summarize the following notations to define the problem.

Table 1: Notations Used in Auto-Scaling Algorithm

Notation	Meaning
J(i)	The job class i
JJ (i)	An job instance of class i
CJ(i)	The number of job instances of class i
V	All VM instance types provided
V (i)	VM of type i
VV (i)	An VM instance of type i
CV (i)	The number of VM instances of type i
J (i) T (j)	Task T (j) in a DAG of job class J (i)
T (JJ (i))	The execution time of JJ (i)
AVG (J (i))	The average execution time of jobs of class J (i)
B (x)	B (h): budget per hour; B (d): budget per day

We use the notations above to define the problem. In proposed algorithm, service and task have the same meaning whereas are used in different context. Each job can be represented using a DAG, a sample for which is shown in Figure 2. A node in DAG represents a task (service) and the edge represents the communication cost required between tasks. DAG must preserve its dependencies at the time of execution, that means the child task cannot start its execution until their parent tasks has completed execution.

Proposed algorithm takes job instances in job queue, VM instances in VM pool, job deadlines, predefined budget, DAG of jobs and historical workload pattern as input, the output is a scaling policy to determine the number of instances of each instance type (Scaling plan = {(VV (i), CV (i))}) and a scheduling policy to determine the instance for each running task at some time point of t (Scheduling plan = { J (i) T (j) →VV (i) | J (i) T (j) ∈J (i)}). The scheduling and scaling policies work together to minimize AVG (J (i)). Proposed algorithm should also consider how to manage budget and define the budget management granularity (monthly, daily, hourly). We finally choose hourly budget management granularity B (h) in proposed algorithm to confirm to the full hour billing model. Hourly budget management granularity makes the budget management

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 13-18 (2014)
Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand

more precise and convenient. The problem is defined in Figure 3.

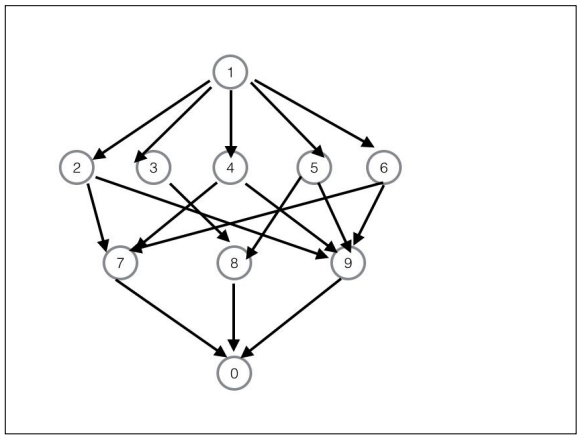

Fig 2: An Example of DAG for Application Jobs

Input: {(JJ (i), CJ (i))}, {(VV (i), CV (i))}, job deadlines, B(h), DAG of jobs, workload pattern.

Output: Scaling policy {(VV (i), CV (i))} and scheduling policy { J (i) $^{T(j)}$ →VV (i) | J (i) $^{T(j)}$ ∈ J (i)}.

Con: $\sum_{} Cost(VV(i)) \times CV(i) \le B(h)$

Fig 3: Problem Definition

B. Proposed Algorithm Based on HEFT

a) Make hourly budget prediction based on the historical workload pattern.

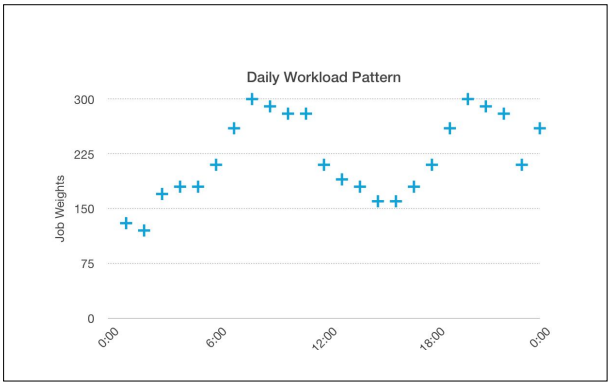

Fig 4: Daily Workload Pattern

The budget should be allocated according to the workload pattern. More budgets are needed to provide more computing power when workload surge happens. We assume that each type of task has a processing weight. The bigger weight means more processing power needed. The job weight is the sum weight of a DAG of tasks shown in Figure 2. The workload pattern curve between hours of a day and the sum weight of jobs was fitted as a mathematical function to offer the basis for calculating the optimal hourly budget. We assume the mathematical function for the curve between hours of a day and the sum weight of jobs is $y = f(h)$, in which h stands for 24 hours of a day while y stands for the jobs' weights of hour h. An example of daily workload pattern curve is shown in Figure 4.

The hourly budget shares the same percentage of the daily budget as the percentage of weight of the day.

$$B(h) = \frac{f(h)}{\sum_{h=0}^{0 \le h \le 23} f(h)} \times B(d) \quad (1)$$

b) Initialize task priority of a DAG of jobs.

Task priority is used in our HEFT based algorithm. Priority is based on mean computation and mean communication costs. For each node in Figure 2 (a DAG of jobs), it should be assigned with a priority value. The increasing order of priority values provides a topological order of tasks. The tasks in task queue should be scheduled in sequence according to the topological order. We use the approach of assigning priority same as HEFT [5] algorithm and the tasks' priorities in this paper are based on downward ranking. Following notations are used in the below task priority initialization algorithm.

Table 2: Notations Used in Task Prioritizing Phase

Notation	Meaning
T (i)	Current task (successor node in DAG)
PT (i)	Dependent task (predecessor node of T (i) in DAG)
P (i)	Priority of task T (i)
P (PT (i))	Priority of PT (i)
AEET (i)	Average estimated execution time of task T (i)
L	The bandwidth of the network link by which predecessor node sends data to successor node.
CC (j, i)	Communication cost of edge (j, i)
Nj	Threshold of number of jobs for hourly execution

Following is the approach for calculating priority of tasks.

1. Create a list of tasks T = {T1, T2, T3, ...T n}
2. Pick task T (i).
   1) If priority P (i) is not set then calculate priority, else pick the next task from the list
   2) Before calculating the priority P (i) of the task T (i), we check if the task has a dependent task PT (i) whose priority is not set yet, If priority of PT (i) is not set yet then calculate priority of PT (i) else find priority of task T (i)
   3) Once all the parameters are calculated, set priority: P (i) = AEET (i) + CC (j, i) + P (PT (i)); Where CC (j, i) equals output data generated by PT (i) divided by L (output / L)

It can be easily shown that the increasing order of priority values provides a topological order of tasks, which is a linear order that preserve the precedence constraints. All the priority information will be recorded in status table and will be updated periodically.

c) Choose the best VM of job for each job class.

In this phase, we define a new notation BV (V) for best VM in which V is a collection of available VM instances {VV1, VV2, … VVn}. BV (V) of task is the chosen VM instance from V for the current task based on HEFT and BV (V) of job is the VM instance in V that is chosen as BV (V) of task for most of tasks in a DAG of this job. The below pseudo-code describes how to calculate BV (V) for a job class.

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 13-18 (2014)
Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand

1. Initialize a job DAG of tasks T = {T1, T2, T3, ...T n} and a VM pool with all VM types V = {V1, V2, V3, ...V n}
2. Schedule tasks to VMs in the VM pool based on the priority of tasks calculated before using HEFT algorithm. We finally get a scheduling plan which can minimize the makespan of job execution. Count the total times of each VM chosen as BV (V) of task. Use a map data structure {VM (i)->times} to record mapping information between VM type and times of VM chosen as the best VM
3. Choose the VM type which has the largest number of times in the Mapping data structure as the best VM for the job

Job dispatcher will dispatch jobs to different job type queue which shown in Figure 1. In proposed algorithm, the jobs with the same BV (V) will be categorized as the same type. When considering whether or not to shut down a VM, we will predicate the number of incoming job types which use the current VM as the best VM based on information provided by step a). If the number exceeds the predefined threshold Nj, which means the current VM is urgently needed in the following hour, we will not shut down the current VM and vice versa. In Figure 5, VM0 is the best VM for job {T0, T1, T6, T7} and also the best VM for job {T0, T3, T6, T7}.

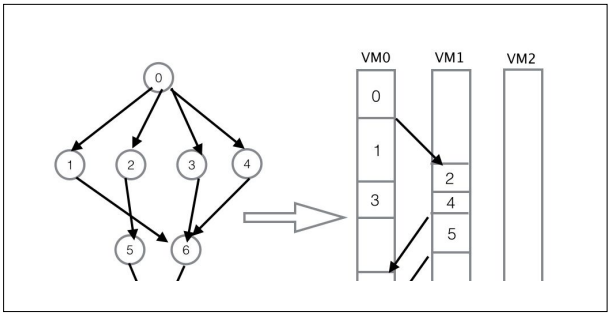

Fig 5: The Schedule Plan Derived based on HEFT

d) Make auto-scaling plan based on HEFT.

Proposed algorithm makes the auto-scaling plan based on BV (V) of the incoming jobs. To get the best performance of the application, proposed algorithm try its best to make sure the tasks run on BV (V) and get the shortest response time for the whole job processing. Budget constraint is a main factor considered in allocating resources through the process of auto-scaling.

In addition to the previous notations used in Table1 and Table2, following are the new notations used in this phase.

Table 3: Notations Used in Auto-scaling algorithm

Notation	Meaning
M	Makespan of current job
D	Deadline of current job
d (i)	Sub-deadline of task T (i)
e (i)	Execution time of task T (i)
EFT (i)	Estimated Finish Time of task T (i)
ED (i)	EFT (i) – d (i)

Following is the algorithm for VM scaling-up.

1. Use a task list of a job DAG {T1, T2, T3, T4, T n} and an available VM instances list VL1 {VV1, VV2, VV3, VV m} as input for proposed algorithm, the VM instances in the VM pool may share the same VM type
2. Calculate the d (i) for each task in the task list:

Sub-deadline for the task = percentage of share of task in application * Deadline of the application + deadline of task's predecessor

3. Pick a task in the task list based on the task priority. For i=1 to n tasks and j=1 to m VM instances, Choose the BV (V = VL1) for the current task for execution
4. Repeat step 3 till we have no unscheduled tasks. Till now, we have a schedule plan for the current job and can get a minimal makespan for this job. Schedule the tasks based on the schedule plan
5. Compare the job's M with the job's D. No matter M>D or M=D or M<D, we have to schedule the current job based on the schedule plan of step 4 first. Because we actually cannot wait that long time of the VM startup which will spend more than 5 minutes for current job scheduling. For this step, if M>D, which means this type of job of current time t cannot be finished within the deadline and we have to scale up more VM instances to meet the job's D. If M<D, which means the job can be finished within deadline, but we still have to figure out whether or not there is a space for the performance improvement
6. Predict the number of jobs with the same BV (V) as the current job for the next hour. If the number of predicted jobs exceeds the threshold Nj, which means more jobs of the type is coming in the next hour, go to step 7, else go to step 1 for another job execution
7. Find all the tasks {TT1, TT2, TT3, TT n} that cannot be finished within sub-deadline and order the tasks by ED (i) decreasingly
8. Use the tasks list {TT1, TT2, TT3, TT n} and VM type list VL2 {V1, V2, V3, V4, V n} as input. Notice that VM type list here contains all the VM types the cloud provider provides. Initialize a scaling VM pool {} as null.
9. Pick a task in the task list and find VV (i) of type V (i) as BV (V' = VL2) for the current task with the constraint cost/hour of VV (i) < B (h). If VV (i)>B (h), then repeat this step till we find a reasonable VV (i) as BV (V')
10. If BV (V') ≠ BV (V) in step 3, which means the current VM pool doesn't contain a VM instance of type V (i). We put one VV (i) {VV (i), 1} in it for after VM scaling. If BV (V') = BV (V), which means the current VM pool already has the best VM for the task, the reason for execution time exceeding sub-deadline may be too tasks waiting in the service queue. That means we have to scale up more instances to meet with the sub-deadline of tasks. The number of VV (i) = length of task queue * e (i) / d (i) – 1. Compute the VV (i) number under the constraint of cost/hour (VV (i)) * CV (i) < B (h). If the requirement can't be met, CV (i) should be decreased by one recursively till it meets this budget constraint requirement.
11. Repeat from step 9 to step 10 till all tasks executed.

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 13-18 (2014)
Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand

12. Finally we get the auto-scaling plan {(VV (i), CV (i))} and scale up the VM instances based on this plan.

Scaling-down decisions are based on some basic rules and typical situations. According to our previous experimental data, one simple rule for scaling-down policy can be described as "shut down the instance when the average CPU is below 10% for more than 10 minutes." However, before shutting down the instance, we have to make sure the current instances are enough for handling the incoming jobs. Back worker in the proposed architecture also shows capability to consolidate VM instances to reduce task concurrency. The scaling-down policy should also consider whether or not the number of incoming jobs using this VM type of the current instance as BV (V) in the next hour exceeds predefined threshold Nj based on prediction algorithm in step B. If it does, scaling-down policy should prepare the VM instance for the incoming job surges in the next hour. Therefore, to shut down a VM instance in the VM pool, we have to make sure all the following requirements are met

1. Time duration for CPU usage percent below 10% > 10 minutes
2. Running time of the VM instance ≈ multiples of hours
3. The number of jobs using this BV (V) for the next hour < Nj

PERFORMANCE ANALYSIS

Proposed algorithm handles with performance-resource mapping problem using auto-scaling approach based on HEFT. In this section, we compare proposed algorithm with other three approaches that are not or partially based on HEFT and the "load vector" based auto-scaling approach proposed in [11]. The approaches are shown in the below table.

Table 4: Approaches Used for Performance Analysis

Approach Id	Auto-Scaling	Auto-Scheduling
1	Rule based scaling	Random scheduling
2	Rule based scaling	HEFT based scheduling
3	HEFT based scaling	Random scheduling
4	Approach proposed in [11]	
5	Approach proposed in this paper	

Rule based scaling is based on resource utilization rules like "Add 2 small instances when the average CPU is above 80% for more than 5 minutes". This rule based auto-scaling algorithm is used in key cloud platforms such as Amazon AWS [1]. The VM type of instances here is pre-specified by cloud user. For random scheduling, we simply assign the task to a resource if it is free at the time of scheduling using greedy based algorithm. Neither rule based scaling algorithm nor random scheduling algorithm is based on HEFT.

In Approach 1, auto-scaling phase is based on simple rules and auto-scheduling phase is based on random-scheduling. In Approach 2, auto-scaling phase is based on simple rule while auto-scheduling phase is based on HEFT. In approach 3, auto-scaling phase is based on HEFT and auto-scheduling phase is based on random-scheduling. In Approach 4, Approach proposed in [11] is also here compared with algorithm proposed in this paper. It mainly focuses on minimizing cost and meeting application deadlines in cloud workflows without considering user budget whereas the proposed algorithm focused on minimizing jobs' makespan within user budget. In Approach 5, both the phases are based on HEFT that is used in proposed algorithm. For approaches from one to three, they don't consider cost or budget constraints while allocating resources whereas approach 4 considers cost as the main factor in allocating resources and proposed algorithm in this paper is focused on meeting the budget constraints. After executing all tasks using above approaches, we compare their makespan with deadline and we also compare their cost with the predefined budget.

We initialize three small VM instances (with 512M ram, bandwidth of 1000 mbps and 1 cpu) in the datacenter with ten hosts. We simulate 6 VM types (vm.tiny, vm.small, vm.middle, vm.large, vm.xlarge, vm.super) for cloud provider in Cloudsim and each VM type has different resource configuration. We use a DAG of jobs in Figure 2 as input and each job is composed of tasks that have different execution time on different types of VM instances. The tasks are configured as different cloudlets with different length in CloudSim. We use an average predefined workload pattern for performance analysis which has two surges at 9:00 AM and 8:00 PM which is shown in Figure 4 in which x axis stand for 24 hours of a day while y axis stands for job weights.

From the workload pattern in Figure 4, we can compute the average hourly weight is 220 (sum weights of the day / 24) units. The average number of incoming jobs of one hour for the performance test is 110. We compute an average job with 2 weights (220 weights / 110 jobs) for later budget assignment. Say the deadline for average job is 3200 time units and we use gain approach in [14] to get the most cost-efficient VM instance type for this average job. We assign all tasks of the job to a typical VM instance VV (i), and compute GainWeight as following:

$$GainWeight = \frac{makespan_{before} - makespan_{after}}{cost_{after} - cost_{before}} \quad (2)$$

We select VV (i) with the biggest GainWeight as the most cost-efficient VM instance. Say that the makespan for this job using the VM instance is 1600 time units and VM instance can only execute one task at once. In this case, one VM instance can only handle with two jobs within jobs' deadline therefore the number for this type of VM should be 55 (110 jobs / 2). We assume that cost for VM instance of this type is 10 cost units, the total cost should be 55 * 10 = 550 cost units. Then, we use an average hourly budget with 580 units (B (h) = 580, adequate budget) for one test and average hourly budget with 450 units (B (h)' = 450, inadequate budget) for another test. We compare average makespan of jobs with deadline and compare average hourly cost when executing all tasks with average hourly budget. The results outputted by using CloudSim simulation are shown in Figure 6 and Figure 7. We can see that jobs' makespan and cost using Approach 5 will change with different user budget whereas the other approaches will stay almost the same since they don't consider user budget as a parameter. Approach 5' in Figure 6 and Figure 7 shows the result when testing with B (h)'.

ISSN 2320 -2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.3 , No.11, Pages : 13-18 (2014)
Special Issue of ICCECT 2014 - Held during 01-02 December, 2014,Bangkok, Thailand

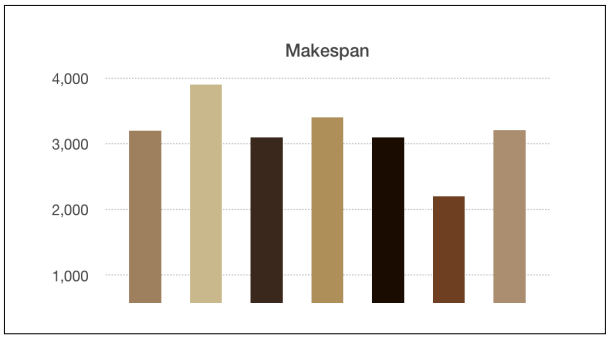

Fig 6: Jobs' Makespan of Five Approaches with Deadline

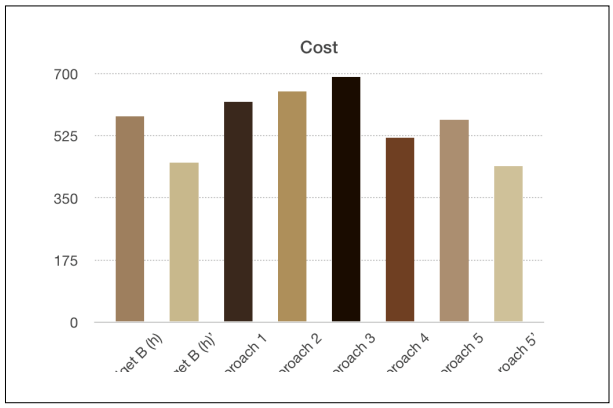

Fig 7: Cost of Five Approaches with Daily Budget

In Figure 6, when using adequate budget B (h), proposed algorithm using HEFT shows the ability to minimize jobs' makespan compared to other approaches and still works well with inadequate budget B (h)'. Approach 2 and approach 3 partially using HEFT has better performance than approach 1 and HEFT based scheduling has bigger impact than HEFT based scaling in respect of shortening jobs' makespan. Approach 4 which uses "load vector" based scaling mechanism and EDF based scheduling algorithm can make sure jobs finished near to deadline with minimum cost. Since it doesn't consider user budget, its performance stays almost the same with different budget parameters.

Figure 7 shows cost variations of five approaches. When using adequate budget B (h), proposed algorithm has cost near to cost of approach 4 with 41% higher performance improvement. But when with inadequate budget B (h)', proposed algorithm will still keep cost under user budget while cost of approach 4 will exceed the user budget. Approach 3 has highest cost because HEFT based auto-scaling algorithm will always find the BV (V) for the current task while the BV (V) cannot be utilized well based on random scheduling algorithm which causes resource waste. Approach 2 will schedule more tasks to the VM which has higher performance and will cause more tasks waiting for execution and scale more random VM instances. Proposed algorithm considers cost as the main factor in allocating resources and will always keep cost under user budget. All the simulations are done using CloudSim simulation toolkit [15].

CONCLUSION

In this paper, we present a mechanism to dynamically scale cloud computing instances based on job deadline and user budget information. Our mechanism optimizes the cloud application performance by scaling up best VM instances in response to dynamic performance behaviors. Budget constraint is a main factor when allocating cloud resources and our mechanism will always keep cost under user budget with different budget possibilities. Evaluation results show that our mechanism can get the best performance of the cloud applications within limited user budget compared to other approaches that are not or partially based on HEFT.

Furthermore, although dynamic workload prediction is not our focus in this paper, effective dynamic workload prediction techniques could handle better with the incoming workload bursting, reduce the effects of delayed instance acquisitions and avoid VM churn. In future we are planning to do more researches on dynamic workload pattern prediction mechanisms and serve the auto-scaling algorithm better in real time.

REFERENCES

[1] Amazon EC2. http://aws.amazon.com/cn/ec2/.
[2] Windows Azure. http://azure.microsoft.com/zh-cn/.
[3] Google AppEngine. https://appengine.google.com.
[4] RightScale. http://rightscale.com.
[5] Topcuoglu, Haluk, Salim Hariri, and Min-you Wu, *Performance-effective and low-complexity task scheduling for heterogeneous computing*, Parallel and Distributed Systems, IEEE Transactions on 13, no. 3 (2002): 260-274.
[6] P. Marshall, K. Keahey and T. Freeman, *Elastic Site Using Clouds to Elastically Extend Site Resources*, 10th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Melbourne, Victoria, Australia, 2010.
[7] [7] M.D. Assuncao, A.D. Costanzo, and R. Buyya, *Evaluating the Cost-benefit of Using Cloud Computing to Extend the Capacity of Clusters*; In Proceedings of the 18th ACM international symposium on High performance distributed computing, Munich, Germany, June 11-13, 2009.
[8] Jiang, Jie Lu, Guangquan Zhang, Guodong Long, "Optimal Cloud Resource Auto-Scaling for Web Applications," DOI 10.1109/CCGrid.2013.73.
[9] D. Menasc and E. Casalicchio, "A Framework for Resource Allocation in Grid Computing," In Proc. of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pp. 259-267, 2004.
[10] J. Yu, R. Buyya and C. Tham, "A Cost-based Scheduling of Scientific Workflow Applications on Utility Grids," In Proceedings of the First IEEE International Conference on e-Science and Grid Computing. Melbourne, Australia, Dec. 2005, pp. 140-147.
[11] Ming Mao, Marty Humphrey, "Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows," SC11, November 12-18, 2011, Seattle, Washington, USA.
[12] Ming Mao, Jie Li, Marty Humphrey, "Cloud Auto-scaling with Deadline and Budget Constraints," 11th IEEE/ACM International Conference on Grid Computing.
[13] Nitish Chopra, Sarbjeet Singh, "HEFT based Workflow Scheduling Algorithm for Cost Optimization within Deadline in Hybrid Clouds," 4th ICCCNT 2013 July 4-6, 2013, Tiruchengode, India.
[14] R. Sakellariou and H. Zhao, "Scheduling Workflows with Budget Constraints," Integrated Research in Grid Computing. CoreGrid series, Springer-Verlag, 2005.
[15] Calheiros, Rodrigo N, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience 41, no. 1 (2011): 23-50.