View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by St Andrews Research Repository

Algorithms for Optimising Heterogeneous Cloud Virtual Machine Clusters

Long Thai
School of Computer Science
University of St Andrews
Fife, UK
Email: ltt2@st-andrews.ac.uk

Blesson Varghese
School of EEE and Computer Science
Queens University Belfast
Belfast, United Kingdom
Email: varghese@qub.ac.uk

Adam Barker
School of Computer Science
University of St Andrews
Fife, UK
Email: adam.barker@st-andrews.ac.uk

***Abstract*—It is challenging to execute an application in a heterogeneous cloud cluster, which consists of multiple types of virtual machines with different performance capabilities and prices. This paper aims to mitigate this challenge by proposing a scheduling mechanism to optimise the execution of Bag-of-Task jobs on a heterogeneous cloud cluster. The proposed scheduler considers two approaches to select suitable cloud resources for executing a user application while satisfying pre-defined Service Level Objectives (SLOs) both in terms of execution deadline and minimising monetary cost. Additionally, a mechanism for dynamic re-assignment of jobs during execution is presented to resolve potential violation of SLOs.**

Experimental studies are performed both in simulation and on a public cloud using real-world applications. The results highlight that our scheduling approaches result in cost saving of up to 31% in comparison to naive approaches that only employ a single type of virtual machine in a homogeneous cluster. Dynamic reassignment completely prevents deadline violation in the best-case and reduces deadline violations by 95% in the worst-case scenario.

I. Introduction

With the advent of cloud computing, users can use on-demand resources offered by cloud providers to build clusters, defined as *cloud virtual machine (VM) cluster*. Such clusters can be dynamically reconfigured by adding/removing resources, such as VMs, in order to accommodate the demands of the workload and to achieve desired performance. However, this task can be challenging since cloud providers offer a wide variety of VM types with different performance capabilities. This challenge is further complicated by the monetary cost incurred by renting VMs.

This paper aims to address the challenge *of building a heterogeneous cloud VM cluster by determining the number and type of VMs to achieve the desired performance while minimising the incurred monetary cost*. We focus on *Bag-of-Tasks (BoT)* applications, which consist of independent tasks that are widely used by scientific communities [1] and commercial organisations [2]. The desired performance of the application is represented as a *deadline*, a user defined time constraint within which the application needs to complete execution; this is a commonly reported Service Level Objective (SLO) [3].

We address the above challenge in two steps. Firstly, we determine the number of VMs required to execute a newly submitted workload. Secondly, we monitor the actual execution on the VMs and dynamically reallocate the workload to prevent deadline violations.

The first contribution of this paper is the development of two approaches that construct a cost-effective heterogeneous Cloud VM cluster in order to execute BoT applications within user specified deadlines while minimising the overall execution cost. The first approach achieves an optimal solution that has minimum monetary costs, but can take a considerable amount of time. The second approach is faster but generates sub-optimal solutions. Our approaches are compared against existing approaches that use a single instance type for developing a homogeneous cluster. The experimental results show that when using the proposed approaches there is a cost saving of 4% to 31%.

The second contribution is a novel mechanism for dynamic reassignment which detects and resolves potential deadline violations during runtime. Experiments highlight that deadline violations that are likely to occur due to estimation errors can be reduced by at least 95%.

This remainder of this paper is structured as follows. Section II presents related work. Section III proposes a mathematical model for selecting resources for single and multiple jobs. Mechanisms for handling job submissions are presented in Section IV. Section V proposes a dynamic assignment mechanism. Our research is evaluated in section VI using simulation and real-world jobs on the cloud. Section VII concludes this paper by considering future work.

II. Related Work

Optimising heterogeneous cloud VM clusters based on objectives, such as a user defined deadline for executing a workload or minimising cost of execution when multiple VMs with varying performance are available, has been investigated by the research community due to the popularity of both the application and resource models.

There is research that has focused on *maximising performance while minimising cost of executing BoT application on the cloud* [4], [5], [6]. These approaches require the trade-off between performance and cost to be represented as a numerical value which is often not easy to be calculated as it needs to in turn consider two different metrics.

Scheduling multiple BoT applications on the cloud so that *both deadline and budget constraints can be satisfied* has been proposed [7]. However, the focus is on resource selection to determine the number of VMs of each type under the assumption that workload distributions are known in advance. On the other hand, our proposed approaches support both resource selection and workload allocation without making an assumption on the workload distribution.

Since computing on cloud resources incurs costs there is a need to *satisfy the given budget constraint while maximising performance* [8] [9]. In this paper, we argue that performance constraints, such as the execution deadline, which represents the desired performance that users want to achieve and is directly related to user experience is equally important as budget constraints. In existing research, we also note users are only able to schedule a single job, which is not the case in the research presented in this paper.

Finally, there is research in optimising the execution of BoT applications on the cloud focusing on *satisfying deadline constraint while minimising the cost*, which is also the objective of this paper. Bossche et al. [10] proposed to offload workload from private cloud to public cloud when necessary in order to satisfy execution deadlines. However, the proposed approach only supported a single application. Menache et al. [11] proposed an approach to calculate the number of on-demand and spot instances in order to execute all BoT jobs within their deadline while reducing the cost. However, the authors only considered one type of instance type, i.e. homogeneous cloud VM cluster, which does in reality harness the potential of building clusters with a wide variety of options provided by cloud providers. In our previous work [12], we presented an approach for scheduling BoT jobs with a deadline on the cloud. This paper significantly extends our previous work by proposing different scheduling approaches, investigating the trade-off between the solving time and a solution's optimality, and performing extensive experiments to present the benefits of using heterogeneous cloud VM cluster instead over homogeneous clusters. This paper also highlights the benefit of dynamic reassignment.

III. Resource Selection Model

In this section, we propose two models to address the problem of i) determining the number and type of VMs in a heterogeneous Cloud VM cluster consisting of instances of different types, and ii) allocating workload to each VM.

A. Environment Model

Let $IT = \{it_1...it_m\}$ be the list of instance types, which are the prototypes to create instances or virtual machines (VMs). Cost per hour of an instance type is denoted as c_{it}.

Let $A = \{a_1...a_n\}$ denote the list of applications executed on the cloud. In this paper, we assume prior knowledge of all applications based on the fact that most applications executed in data centres are recurring [3], [13]. Task execution time $e_{a,it}$ is *the average time (in seconds) to execute one task of an application a on an instance of type it.*

Let J denote the list of jobs to be scheduled. Each job j belongs to an application a_j and contains a number of tasks n_j. Its deadline is d_j. We assume prior knowledge of applications but not individual jobs; we know which applications are executed but not when they are executed and the number of tasks.

Let V denote the list of instances or VMs. An instance type of an instance v is denoted as it_v . As each instance needs a certain amount of time to be booted, let β be the average boot time. Let $e_{j,v}$ be *the amount of time that an instance v executes tasks of a job j*, which also means that $e_{j,v} = 0$ if v does not execute any tasks of j.

Let r_v be the running time of an instance v, which can be calculated by adding the sum of the execution times of all jobs on v with the average boot time. However, r_v is 0 if it does not execute any task at all:

$$r_v = \begin{cases} \beta + \sum_{j \in J} e_{j,v}, & \text{if } \sum_{j \in J} e_{j,v} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

B. Multiple and Single Job Approaches

We propose two approaches to determine the number of VMs within a cloud VM cluster for executing the job(s). The first approach aims to find the optimal solution for all submitted jobs while the second approach achieves a local optimal for each job.

1) Approach 1: Resource Selection for Multiple Jobs: First, let X be a list of binary values which indicate whether instances are created or not. In other words, for $v \in V$, $x_v = 1$ if an instance is created and a user has to pay for it. Otherwise, $x_v = 0$.

We assume that execution of jobs on an instance is ordered in a similar manner to how jobs are ordered within J. For instance, if j_1 is ordered before j_2 in J, then j_1 must be executed before j_2 on all instances. In this paper, we use a pre-defined priority of jobs for ordering.

Since jobs are sequentially executed based on predefined order, on any given instance, the sum of the boot time β and all job execution times cannot exceed the deadline of the last executed job, i.e.:

$$\sum_{j \in J} e_{j,v} \leq d_j - \beta \tag{2}$$

Given the execution time of a job on an instance and the time it takes to execute one task, it is possible to calculate the number of tasks executed on an instance:

$$n_{j,v} = x_v \times \lfloor \frac{e_{j,v}}{e_{j,it_v}} \rfloor \tag{3}$$

The right hand side of the Equation 3 is multiplied with the indicator x since an instance needs to be created in order to execute any tasks. Thus, if an instance v is not created, then the number of tasks of any job executed by v must be 0, since $x_v = 0$. Moreover, a floor function is applied since each task must be fully executed on an instance.

In order to make sure all tasks of a job are executed an additional constraint is imposed which is:

$$\sum_{v \in V} n_{j,v} = n_j \tag{4}$$

The cost of using one instance can be calculated by multiplying the cost of one hour to the actual using time *rounded up to the nearest hour*:

$$c_v = \lceil \frac{r_v \times x_v}{3600} \rceil \times c_{it_v} \tag{5}$$

Notably, as a user only has to pay for instances that are created, i.e. $x = 1$. Hence, the running time of each instance needs to be multiplied to the indicator x.

The total cost is the sum of costs of all instances:

$$COST = \sum_{v \in V} c_v \tag{6}$$

The optimisation problem is presented as follows:

$$\begin{aligned} \text{minimise} \quad & COST = \sum_{v \in V} c_v \\ \text{subject to} \quad & \sum_{j \in J} e_{j,v} \leq d_j - \beta \\ & \sum_{v \in V} n_{j,V} = n_j \end{aligned} \tag{7}$$

2) Approach 2: Resource Selection for Single Job: Instead of selecting resources for executing multiple jobs, in the second approach resources are selected for executing one job, which is hypothesised to be faster, since it aims to find a local optimal over a global optimal solution. However, since only one job is considered at a time, the solution may not be optimal. The comparison between the two approaches is presented in a later section.

Let $n_{j,it}$ be the number of tasks of job j that one instance of type it can execute before a deadline:

$$n_{j,it} = \lfloor \frac{d_j - \beta}{e_{j,it}} \rfloor \tag{8}$$

Let ni_{it} be the number of instances of type it. The total number of tasks executed by instances of type it are $ni_{it} \times n_{j,it}$. The following constraint is used to execute all tasks of a job before its deadline:

$$n_j = \sum_{it \in IT} (ni_{it} \times n_{j,it}) \tag{9}$$

The problem of selecting resource for executing one job within its deadline is modelled as:

$$\begin{aligned} \text{minimise} \quad & COST = \sum_{v \in V} c_v \\ \text{subject to} \quad & n_j = \sum_{it \in IT} (ni_{it} \times n_{j,it}) \end{aligned} \tag{10}$$

The above model is solved for each job since it finds the resource to only execute one job.

IV. Handling Job(s)

The submission handling process selects the optimal number of resources required for the jobs and assigns tasks to each instance such that they are executed within deadlines.

There may be existing VMs when a batch of jobs arrives. It is possible for existing VMs to execute new tasks if *it does not result in deadline violations* (the execution of a job finishes after its deadline). If existing VMs can execute all tasks of the newly submitted jobs, then no additional VMs are required. The assignment process is presented by Algorithm 1 which loops through each job and given instance. First, the finish time of an instance is calculated (Line 4). The finish time on an instance is when it has finished executed assigned tasks and is ready to start executing new tasks.

Then, the execution time of a job on an instance is the difference between its deadline and an instance's finish time (Line 5). If the execution time is positive, some tasks of a job to an instance (from Line 6 to Line 8). It should be noted that we try not to prolong an instance's finish time. The reason is that instead of extending an instance's finish time for another hour, and pay for it, we can just create a new one with the same type.

Notably, Algorithm 1 tries to assign as much tasks as possible to the given list of existing instances. As a result, it does not guarantee that all tasks will be assigned. New instances need to be created in order to accommodate tasks which cannot be assigned to the existing VMs.

Algorithm 1 Assignment

```
1: function ASSIGN($J, V$)
2:     for $j \in J$ do
3:         for $v \in V$ do
4:             $fi_v \leftarrow$ finish time of v
5:             $e_{j,v} \leftarrow d_j - fi_v$
6:             if $e_{j,v} > 0$ then
7:                 $n_{j,v} \leftarrow \frac{e_{j,v}}{e_{a_j,it_v}}$
8:                 Assign $n_{j,v}$ of $j$ to $v$
```

We propose two algorithms for handling job submissions corresponding to two approaches presented in Section III-B.

A. Approach 1: Assigning Resources for Multi Jobs

Algorithm 2 uses the model shown in Equation 7 to select resources for the submitted jobs.

As shown in *Line 2*, jobs are firstly assigned to a list of existing instances, denoted as V_e.

If there are any jobs with tasks left, then a list of new resources denoted as V_n are acquired by solving Model 7 as shown in *Line 4*. Finally, new instances are added to the list of existing instances in *Line 5*.

Algorithm 2 Submission Handling

```
1: function SUBMISSION_HANDLING($J, V_e$)
2:     $ASSIGN(J, V_e)$
3:     if There are jobs with remaining tasks in $J$ then
4:         $V_n \leftarrow$ solution of Model 7
5:         $V_e \leftarrow V_e \cup V_n$
```

B. Approach 2: Assigning Resources to Single Job

Algorithm 3 Submission Handling

```
1: function SUBMISSION_HANDLING($J, V_e$)
2:     for $j \in J$ do
3:         $ASSIGN(j, V_e)$
4:         if There are remaining tasks in $j$ then
5:             $V_n \leftarrow$ solution of Model 10
6:             $ASSIGN(v, V_n)$
7:             $V_e \leftarrow V_e \cup V_n$
```

Algorithm 3 presents the selection of resources when only one job at a time is considered. It is an iterative process which performs the following steps for each job: (i) assigning tasks to existing instances (Line 3) and (ii) creating and assigning remaining tasks to new instances if necessary (Line 5 and Line 6). New instances created to execute a job are added to the list of existing instances (*Line 7*).

V. Dynamic Reassignment

In the previous section, static scheduling approaches were presented that are employed before the actual execution based on the estimated performance of VMs and jobs, such as task execution time. However, runtime performance is not guaranteed and it is likely that there may be tasks that take more (or less) time to be executed than estimated. This performance variation is not considered when handling submissions and can result in unexpected delays which lead to deadline violations. The delay of one job on an instance leads to a cascading effect whereby all jobs scheduled on that instance may be delayed.

In this section, we propose a dynamic monitoring and reassignment mechanism, which aims to prevent deadline violations by moving tasks from an instance that cannot meet the specified deadline to another one that can accommodate extra workload without a risk of deadline violation.

A. Monitoring Progress

Monitoring is periodically performed to retrieve the progress information of instances. This information for each instance contains the currently executed job and the number of tasks that have already executed.

Algorithm 4 Dynamic Reassignment

```
1: function REASSIGNMENT($V$)
2:     $V_g \leftarrow \emptyset$
3:     $V_r \leftarrow \emptyset$
4:     for $v \in V$ do
5:         if $v$ is still executing then
6:             $w \leftarrow$current job
7:             $f_e \leftarrow w'$s estimated finish time
8:             if $f_e > d_w$ then
9:                 $V_g \leftarrow V_g \cup \{v\}$
10:        else
11:            if $v$ has no remaining job(s) then
12:                $V_r \leftarrow V_r \cup \{v\}$
13:            else
14:                $st \leftarrow$start time of the next job
15:                if $NOW <$ st then
16:                    $V_r \leftarrow V_r \cup \{v\}$
17:    Order $V_g$ by estimated violating time
18:    Order $V_r$ by allowed receiving time
19:    for $v_g \in V_g$ do
20:        for $v_r \in V_r$ do
21:            $t_r \leftarrow$ receiving time of $v_r$
22:            $n_r \leftarrow \lfloor \frac{t_r}{e_{v_r,a_w}} \rfloor$
23:            $n_g \leftarrow \lceil \frac{f_{n_g} - d_{n_g}}{e_{v_g,a_w}} \rceil$
24:            $n_r \leftarrow \min(n_g, n_r)$
25:            if $n_r > 0$ then
26:                Move $n_r$ tasks from $v_g$ to $v_r$
27:                $V_r \leftarrow V_r \setminus \{v_r\}$
```

B. Dynamic Reassignment Algorithm

Algorithm 4 presents the dynamic reassignment process, which takes place after the monitoring process and consists of two parts.

The first part (from Line 2 to Line 16) aims to create two lists: (i) of potentially violating instances and (ii) of recipient instances that can receive more tasks. Potentially violating instances are those whose estimated finish times are after the deadlines (from Lines 6 to Line 9). On the other hand, a recipient instance is either idle or has finished current execution but the start time of the next execution is in the future (Lines 11 and 16. A recipient instance can execute extra tasks from the current time to the expected start time of the next job.

In the second part, tasks are moved from violating instances to recipient instances; the sets of violating and recipient instances are ordered based on violation time and allowed receiving time respectively (Lines 17 and 18). The violation time is calculated as the difference between an estimated finish time and a deadline. The allowed receiving time is the amount of time from the current time to the minimum value of (i) a deadline, (ii) a start time of the next job (if an instance is not idle), and (iii) the termination time (if an instance is idle).

For each violating instance, the number of tasks that need to be moved is calculated based on the violating time, which is the difference between estimated finish time and the deadline, and the task execution time (Line 23). The number of tasks a recipient instance can receive is calculated based on its allowed receiving time (Line 22). If an instance can receive tasks from a violating instance, then reassignment is performed (Line 25 and Line 26).

After receiving tasks, the recipient instance is removed from the list, since it cannot receive more tasks (Line 27). This is done in order to avoid aggressive reassignment of tasks from many instances onto an instance.

It should be noted that Algorithm 4 does not guarantee complete resolution of potential deadline violations, i.e. when there are not enough recipient instances to receive tasks from violating ones. This may be resolved by adding additional instances, which requires further investigation and will be reported elsewhere.

VI. Evaluation

This section compares the two proposed approaches against others that use the same type of instance in a homogeneous cloud cluster. We use Gurobi[1], a commercial mathematical programming environment to solve the models represented by Equation 7 and Equation 10.

Our evaluation framework consists of a centralised master which schedules BoT job execution on the cloud, periodically monitors and performs dynamic reassignment.

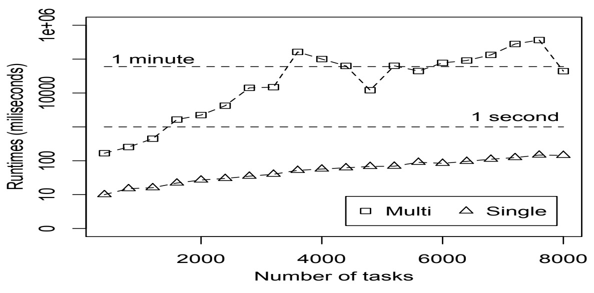
A. Submission Handling Evaluation

In this section, experiments were performed to compare the single and multi-job submission approaches presented in Section IV. The effect of four environmental and workload factors, namely the number of tasks, jobs, instance types and deadlines, on two metrics - (i) runtime of an approach to find a solution, and (ii) monetary cost for executing the submitted jobs - were considered. Four sets of experiment were performed to evaluate the effect of the above four factors on the runtime and cost metrics. All sets of experiment started with 3 instance types, 3 jobs, each comprising 100 tasks, and an average deadline of 1000 seconds.

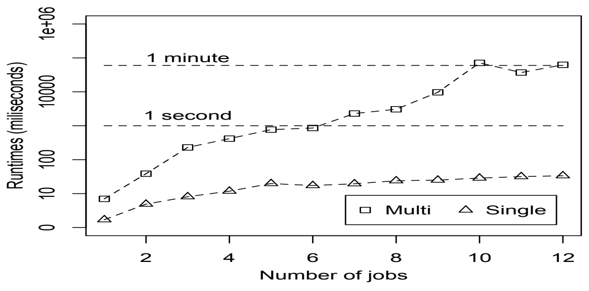
1) Comparing Runtime: Figure 1 presents the solving time using single-job and multi-job approaches in different scenarios. In all cases, the runtime of the single-job approach is below one second and lower than the multi-job approach.

Figure 1a and Figure 1b shows that as the number of tasks and jobs increase, the runtime of the multi-job approach increases to more than a minute. This is because as the workload size increased, more resources were required to meet the increased workload demand. Consequently, the

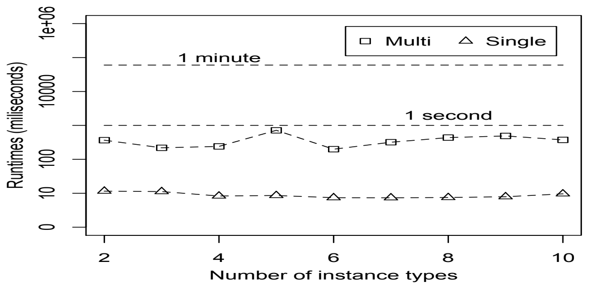
[1]http://www.gurobi.com



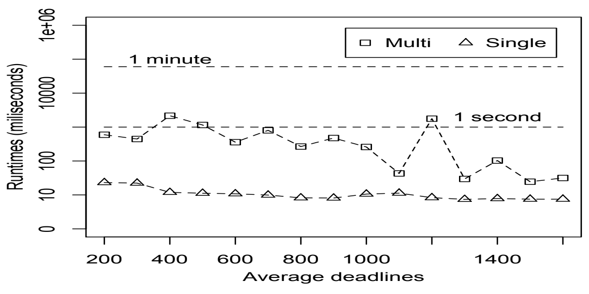
(a) Varying number of tasks



(b) Varying number of jobs



(c) Varying number of instance types



(d) Varying job deadlines

Figure 1: Runtime of multi-job and single-job submission approaches

multi-job approach needs to traverse through a larger search space than the single job approach requiring more time. Hence, the multi-job approach may not be ideal if a fast scheduling decision is required given a large workload.

Occasionally, the multi-job approach was able to find a solution faster when a workload increased (refer Figure 1a). We believe this is because the Gurobi solver was able to find a quick path to traverse the search space and find a solution. However, the general trend is that the solving time increases as the workload increases.

The number of instance types did not noticeably affect the runtime of both approaches as shown in Figure 1c. This can be explained as the Gurobi solver found a subset of the most cost effective instances, thereby reducing the search space without requiring to consider all possible instance types.

Figure 1d shows that as the deadline for a job was increased (minimising the urgency to be completed) the runtime of the approaches decreased. This is because fewer instances were required when a deadline was increased since more tasks could be assigned to an instance. Consequently, the search space is smaller, thus reduced runtimes.

2) Comparing Cost: Figure 2 shows cost saving that can be achieved by using the multi-job approach in comparison to the single-job approach. It is inferred that the multi-job approach was a cheaper solution most of the time and achieves savings up to 6%. This is because the multi-job approach found the resources suited for all the jobs which was a global optimal. On the other hand, the single job approach found the best set of instances for one job at a time which was a local optimal. However, there were a few cases in which the multi-job approach offered no cost savings as both approaches provided the same scheduling decision.

B. Amazon Cloud-based Experiments

1) Experimental Setup: Experiments were performed on the Amazon Web Service (AWS) cloud to compare the proposed approaches on real-world applications in order to evaluate both submission handling and dynamic reassignment. For the experiments reported in this paper three instances shown in Table I were selected.

Three real-world applications with different workload characteristics were employed. The first is a Molecular Dynamics Simulation (MDS) of a 250 particle system in which the trajectory of the particles and the forces they exert are solved using a system of differential equations [14]. MDS is embarrassingly parallel and CPU intensive. The second one uses SVM^{light2} to classify data sets provided as input files ranging from $100MB$ to $500MB$. This application only uses one core on a machine. The third one uses $lbzip2^3$, a parallel compression utility, to compress files ranging from $500MB$ to $1GB$. It supports multiple CPU cores which communicate with each other.

Prior to the experiment, a sampling process to generate the average task execution time of all applications on the instance types was performed as shown in Figure 3. The results indicate that MDS benefited from parallel execution; increasing the number of CPU cores resulted in significant performance improvement. The communication overhead between the parallel cores in $lbzip2$ degrades performance. There is minimal gain for SVM^{light} since it relies on sequential execution.

2) Evaluated Approaches: The following approaches were evaluated: (i) single-job resource selection with (*single.dyna*) and without (*single.nodyna*) dynamic reassignment, (ii) multi-job resource selection with (*multi.dyna*) and without (*multi.nodyna*) dynamic reassignment.

The proposed approaches were compared to commonly used approaches that only use one instance type to create a homogeneous cloud VM cluster. As mentioned earlier, $n_{j,it}$ is the number of tasks of job j that one instance of type it

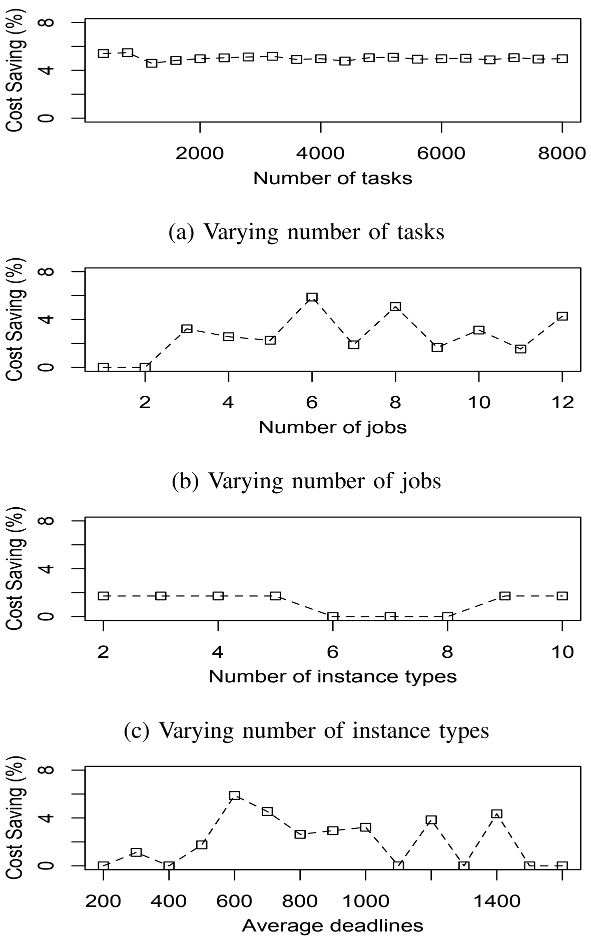


Figure 2: Cost comparison of multi-job and single-job approaches

Table I: AWS Instance Types

Name	vCPU	ECU	Mem	Storage	Price
m3.medium	1	3	3.75	4	\$0.073
m3.large	2	6.5	7.5	32	\$0.146
m3.xlarge	4	13	15	80	\$0.293

[2]http://svmlight.joachims.org/
[3]http://lbzip2.org/

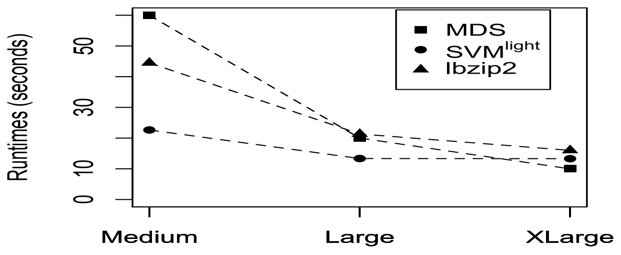


Figure 3: Task execution time

can execute before a deadline. Hence, the number of VM of it required to execute all tasks of j is $\frac{n_j}{n_{j,it}}$.

In total, there are 6 different homogeneous approaches resulting from 3 instance types and the support for dynamic reassignment: *medium.dyna, medium.nodyna, large.dyna, large.nodyna, xlarge.dyna, and xlarge.nodyna*.

The jobs were submitted in two batches. The first batch was submitted at the beginning of the experiment, comprising 3 jobs corresponding to each application, and each job consisted of 100 tasks requiring to meet a deadline of 1200 seconds. The second batch also had 3 jobs, each of which had the same deadlines as the former batch but had 150 tasks, was submitted 300 seconds after experiment started. Finally, based on the sampling experiment, the booting time β, was set to 100 seconds.

3) Cost Comparison: Figure 4 shows the total cost incurred from using different approaches. In general, using heterogeneous cloud VM clusters achieved lower cost in comparison to use homogeneous ones. The cost saving ranges from 4% to 31%. The multi-job approach found a cheaper solution ($1.606) in comparison to the single-job approach ($1.679).

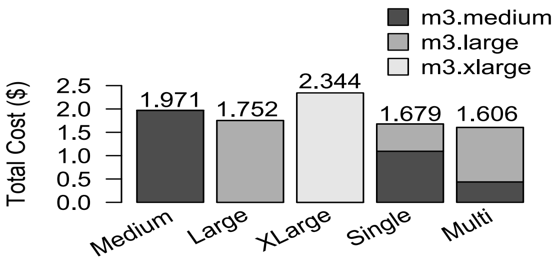


Figure 4: Cost incurred for each approaches

Both approaches did not employ instances of m3.xlarge type. The single-job approach tended to use more instances of m3.medium type and fewer instances of m3.large type (5 m3.medium and 4 m3.large instances) compared to the multi-job approach (6 m3.medium and 8 m3.large instances).

Figure 5 shows the assignment ratio between jobs and instance types, which is the amount of workload of the job

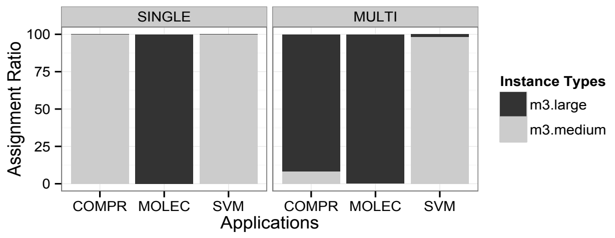


Figure 5: Assignment ratio of applications on instances

assigned to each instance type. For instance, the single-job resource selection approach assigned all tasks of SVM^{light} and $lbzip2$ to m3.medium instances and MDS to m3.large instances. The reason is that only one job at a time was considered and the most cost effective instance type for executing each job was selected. On the other hand, the multi-job resource selection approach distributed a significant proportion of $lbzip2$ and MDS and some of SVM^{light} workload to m3.large instances. Here all jobs were taken into account and the combination of resources most cost effective for all jobs were selected.

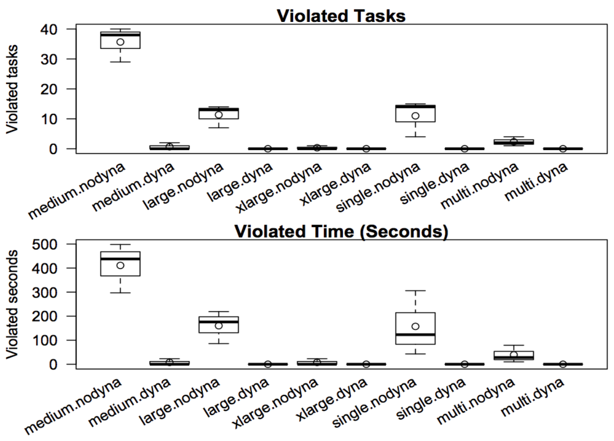


Figure 6: Deadline violation of the four approaches

4) Violation Comparison: In this research, we measure violation as (i) the number of tasks finishing after their job deadlines and (ii) the delay of all job executions, which is the difference in seconds between the finish times of violating jobs and their deadlines.

Figure 6 presents the violation of all approaches for the batch of jobs considered in the above section. In is evident that dynamic reassignment greatly reduces the violation for all approaches. In fact, there was only one case in which violation happened in the homogeneous cluster of medium VMs. However, the violation time was 23 seconds, which was 95% lower compared to 438 seconds when dynamic reassignment was not used.

Without dynamic reassignment, the multi-job resource selection approach resulted in lower execution time when compared to the single-job resource selection approach. This can be explained by Figure 7 highlighting the average total execution time (from when the first job is submitted until the finish time of the last job) of each approach. Using the multi-job resource selection approach resulted in lower total execution time in comparison to the single-job approach thereby reducing the chances of violation.

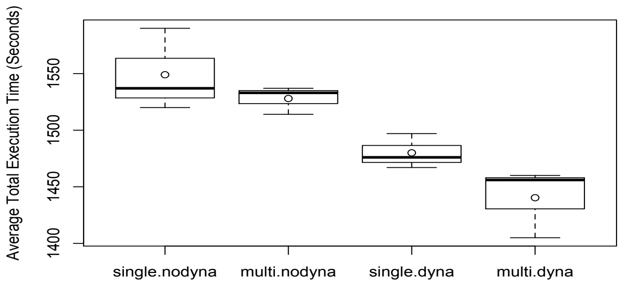


Figure 7: Total execution time using the four approaches

VII. Conclusions and Future Work

This paper aimed to address the problem of cost-effectively building a heterogeneous cloud VM cluster to execute BoT jobs within a user specified deadline. Two approaches were proposed for handling multiple and single jobs. In the first approach, all jobs were taken into account for deriving a scheduling decision. In the second approach, one job was considered at a time and scheduled onto a cloud instance. Experimental studies evaluating the two approaches highlighted that the multi-job approach reduced both the total monetary cost and the overall execution time by up to 6%. The single job approach managed to take less than one second for making a scheduling decision for all submitted jobs. The single-job approach is suitable for a system that handles a large workload while the multi-job approach can maximise cost savings on the cloud. When compared to naive approaches the proposed approach is able to achieve cost savings from 4% to 31%. A dynamic re-assignment mechanism was proposed and developed to handle unexpected delays during the execution of the BoT application. Our evaluation highlighted that violation of deadlines was greatly reduced using the proposed mechanism. In the worst case scenario, using dynamic reassignment the approach reduced deadline violation by 95% and no deadlines were violated in the best case.

In the future, we will investigate how additional instances can be added to the cluster during dynamic re-assignment in order to completely resolve potential violation. The effect of the order of jobs in scheduling will be investigated.

Acknowledgment

This research was supported by an Amazon Web Services Education Research grant.

References

[1] A. Iosup and D. Epema, “Grid Computing Workloads,” *IEEE Internet Computing*, vol. 15, no. 2, 2011.

[2] A. Goder, A. Spiridonov, and Y. Wang, “Bistro: Scheduling Data-Parallel Jobs Against Live Production Systems,” in *USENIX Annual Technical Conference*, 2015.

[3] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, “Jockey: Guaranteed Job Latency in Data Parallel Clusters,” in *Proceedings of the ACM European Conference on Computer Systems*, 2012.

[4] R. Duan, R. Prodan, and X. Li, “Multi-Objective Game Theoretic Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, 2014.

[5] M. R. H. Farahabady, Y. C. Lee, and A. Y. Zomaya, “Pareto-Optimal Cloud Bursting,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, 2014.

[6] L. Thai, B. Varghese, and A. Barker, “Executing Bag of Distributed Tasks on the Cloud: Investigating the Trade-Offs between Performance and Cost,” in *IEEE Conference on Cloud Computing Technology and Science*, 2014.

[7] M. Mao, J. Li, and M. Humphrey, “Cloud Auto-Scaling with Deadline and Budget Constraints,” in *IEEE/ACM International Conference on Grid Computing*, 2010.

[8] L. Thai, B. Varghese, and A. Barker, “Task Scheduling on the Cloud with Hard Constraints,” in *IEEE World Congress on Services*, 2015.

[9] A. Oprescu and T. Kielmann, “Bag-of-Tasks Scheduling under Budget Constraints,” in *IEEE International Conference on Cloud Computing Technology and Science*, 2010.

[10] R. Van Den Bossche, K. Vanmechelen, and J. Broeckhove, “Online Cost-efficient Scheduling of Deadline-constrained Workloads on Hybrid Clouds,” *Future Generation Computer Systems*, vol. 29, no. 4, 2013.

[11] I. Menache, O. Shamir, and N. Jain, “On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud,” in *International Conference on Autonomic Computing*, 2014.

[12] L. Thai, B. Varghese, and A. Barker, “Minimising the Execution of Unknown Bag-of-Task Jobs with Deadlines on the Cloud,” in *ACM International Workshop on Data-Intensive Distributed Computing*.

[13] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, “Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing,” in *USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[14] K. Bowers, E. Chow, H. Xu, R. Dror, M. Eastwood, B. Gregersen, J. Klepeis, I. Kolossvary, M. Moraes, F. Sacerdoti, J. Salmon, Y. Shan, and D. Shaw, “Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters,” in *ACM/IEEE Supercomputing Conference*, 2006.