Budget Constrained Execution of Multiple Bag-of-Tasks Applications on the Cloud

Long Thai, Blesson Varghese and Adam Barker
School of Computer Science, University of St Andrews, Fife, UK
Email: {ltt2, varghese, adam.barker }@st-andrews.ac.uk

***Abstract*—Optimising the execution of Bag-of-Tasks (BoT) applications on the cloud is a hard problem due to the trade-offs between performance and monetary cost. The problem can be further complicated when multiple BoT applications need to be executed. In this paper, we propose and implement a heuristic algorithm that schedules tasks of multiple applications onto different cloud virtual machines in order to maximise performance while satisfying a given budget constraint. Current approaches are limited in task scheduling since they place a limit on the number of cloud resources that can be employed by the applications. However, in the proposed algorithm there are no such limits, and in comparison with other approaches, the algorithm on average achieves an improved performance of 10%. The experimental results also highlight that the algorithm yields consistent performance even with low budget constraints which cannot be achieved by competing approaches.**

I. Introduction

Bag-of-Tasks (BoT) is defined as a collection of independent and identical tasks, which can be executed by the same application but in any order. It is possible to split a BoT into sub-BoTs, each of which is assigned to one separate machine for execution. As a result, BoT applications are usually executed in a distributed environment, for instance, they account for more than 75% of Grid computing workloads [1].

With the advent of cloud computing [2] distributed computing resources ranging from basic to compute optimised, or memory optimised machines are available on a pay-as-you pricing scheme. Cloud computing therefore offers a cost-effective solution to execute BoT applications, in which a user is free to choose the type and quantity of resources required for her application.

A key challenge when executing BoT applications on the cloud in order to achieve maximum performance is the trade-off between decreasing the time it takes to execute individual tasks and increasing the number of tasks executed at the same time. Using high-performing (but expensive) machines can reduce the time to execute an individual task. On the other hand, a larger collection of cheaper machines will maximise execution parallelism. An additional challenge is encountered when a user needs to execute multiple BoT applications at the same time, as each application will differ in performance on the same type of machine. For example, tasks of a CPU intensive application will perform best on a compute optimised machine; a memory optimised machine may not be best suited.

In this paper, a heuristic algorithm which considers the diversity in cost, machine types and application performance is proposed to solve the problem of executing multiple BoT applications on the cloud given a user's budget constraint. The algorithm efficiently assigns tasks to cloud machines of different types such that the budget constraint is not violated while minimising the execution time. The algorithm is evaluated against existing approaches and achieves better performance for a given budget.

The research contributions of this paper are as follows: (i) a mathematical model of the problem of executing BoT on the cloud while taking into account a budget constraint, ii) the development and implementation of a heuristic algorithm that aims to maximise the performance of a BoT on the cloud while satisfying the given constraint, and iii) an evaluation which compares the proposed algorithm with other approaches.

The remainder of this paper is organised as follows. Section II considers research related to that presented in this paper. Section III presents a mathematical model of the problem. Section IV proposes the algorithm for executing multiple BoT applications. Section V evaluates the proposed algorithm. Section 6 concludes this paper by considering future work.

II. Related Work

One popular framework for executing BoT is BOINC [3], which distributes tasks to resources whose computation is volunteered from around the world.

The MyGrid [4] framework facilitates the execution of a BoT on the grid by minimising the execution time. This is achieved by replicating and assigning unfinished tasks to idle resources. Task scheduling algorithms have been previously investigated [5]. The location of input data can be taken into account to reduce the execution time of BoT and improve the Quality-of-Service (QoS) [6], [7]. Independent file-sharing tasks can be executed on the grid efficiently by preventing the bottleneck of all machines executing the tasks requiring to download data from the centralised server [8]. Scheduling algorithms in which each task requires data distributed at multiple sources and satisfies both deadline and budget constraints have been considered [9].

Executing multiple BoT applications is also widely investigated by researchers. There are decentralised approaches to increase the throughput and fairness of the execution [10]. Another strategy allows multiple tasks to be executed concurrently on the same machine without severely affecting

performance [11]. An evaluation of different strategies to execute multiple BoT applications on the Grid is considered by Anglano and Canonico[12].

However, those researches in Grid computing may not be applicable to cloud environment as their resources are already available (i.e. machines are already running) and usually free of charge. On the other hand, a cloud user has to decide (and pay for) the type and the number of resources required **before** the actual execution. Furthermore, the applicability of Grid computing is not as wide as cloud research since those platforms are mostly accessible to organisations that can afford to invest into the infrastructure and maintain it.

Recently, researchers have started to focus on employing the cloud for executing BoT. For example, statistical approaches to schedule BoT on the cloud given a budget constraint [13] and approaches to cost-effectively execute BoT on multiple clouds [14] are recent efforts. A comparison of scheduling algorithms for executing multiple BoTs on the cloud has been investigated [15]. Mao et al. propose a approach to scale Cloud resource based on deadline and budget constraints using constraint programming [16]. In our previous work [17], we investigate the execution of a Bag-of-Distributed-Tasks (BoDT) application, in which each task required data from a globally distributed source. Hence, the BoDT application is split into multiple BoT applications, each of which only contains tasks from one data source. Due to the geographical and network distance, the task execution, which includes downloading input data, of tasks from different applications, i.e. data source, can be different. With the same amount of money, a user can obtain either a small number of high performance but expensive machines or many low performance but cheap ones. The trade-off between application makespan and execution parallelism is investigated in [18].

In comparison with [13], [14], [17] which make an assumption that there is a limit to the number of cloud resources, our paper allows a user to acquire as many resources allowed by the budget. Moreover, it performs not only resource provisioning [13], [16], but also task assignment for multiple applications to cloud resources. Even though task assignment is more complicated due to the high number of tasks, it offers a better flexibility and is more suitable for cases when the execution time of each task is not similar due to additional factor such as their data size.

III. Problem Modelling

In this section, the problem of executing multiple BoT applications on the cloud with budget constraint is modelled.

A. System Model

Let M be the number of applications and the set of application be $A = \{A_1 ... A_M\}$. Each application is a collection of the same type of tasks denoted as $A_i = \{t_{i,1} ... t_{i,|A_i|}\}$.

Let $T = \bigcup_{A_i \in A} A_i = \{t_1, t_2, ... t_{\sum_{A_i} |A_i|}\}$ be the list of tasks, thus $|T| = \sum_{A_i \in A} |A_i|$. Each task belongs to one application ($\forall t \in T : \exists! A_j \in A$), such that $t_i \in A_j$. For any given task t, its application can be found as A_t.

Each $t \in T$ is measured by $size_t$, which is used to compare each task **of the same application**. The size of a task can be the actual size of its input data or any parameter related to its complexity; for example, the number of training iteration for a machine learning application. The value of $size_t$ determines the time taken by the application to run on similar hardware; more execution time is required for a larger value.

Let $IT = \{it_1 ... it_N\}$ be the set of N instance types offered by the cloud providers. The cost per hour of an instance is denoted as c_{it}.

The performance of each type of instance changes from one application to another since there are multiple applications. Let P be the performance matrix of size $N \times M$. $P_{i,j}$ is the time in seconds taken by a instance type it_i to process one unit of size of a task of an application A_j. For each instance type $it_i \in IT$, its performance is the vector $P_{it_i} = P_i = \{P_{it_i,A_1} ... P_{it_i,A_M}\}$ corresponding to all applications.

In order to acquire the performance between instance types and applications, we suggest to perform some test runs as, to the best of our knowledge, there is not yet any research in predicting application's performance on different types of machine.

The execution time of a task t using instance type it is $exec_{it,t} = P_{it,A_t} \times size_t$. Thus, the execution time of the collection of T on it can be calculated as $exec_{it,T} = \sum_{t \in T} exec_{it,t}$.

We assume in this model that there is no pair of instance types that have the same performance and cost. So in the model it is possible to have multiple instances with the same either performance or cost.

$$P_{it_i} = P_{it_j} \wedge c_{it_i} = c_{it_j} \iff it_i = it_j \tag{1}$$

The system in which multiple applications must be executed on the cloud consisting different VM types can be represented as (A, IT).

B. Problem Model

The execution plan can be represented as the list of VMs, each of which is created from one instance type and has the list of assigned tasks.

So, assume that $VM = \{vm_1 ...\}$ is the execution plan in which each $vm \in VM$ is created based on one instance type $it \in IT$. For $vm \in VM$, it_{vm} denotes the type of vm. Additionally, VM_{it} be the list of VMs created from the same instance type it.

Let T_{vm} be the list of tasks assigned to $vm \in VM$. The time to execute a task t that is assigned to vm is:

$$exec_{vm,t} = exec_{it_{vm},t} = P_{it_{vm},A_t} \times size_t \tag{2}$$

The following constraint must be satisfied for all tasks to be executed:

$$\bigcup_{vm \in VM} T_{vm} = T \tag{3}$$

Moreover, one task cannot be assigned to multiple VMs and this condition is represented as

$$T_{vm_i} \bigcap T_{vm_j} = \emptyset \text{ if } vm_i \neq vm_j \tag{4}$$

A start up time is required to boot a VM into a usable state and this overhead is denoted as o. The overhead is paid for by the user although a task cannot be executed on the VM during start up.

The execution time of $vm \in VM$ is the sum of the time taken to execute all tasks assigned on the VM and the time for start up denoted as

$$exec_{vm} = o + \sum_{t \in T_{vm}} exec_{vm,t} \tag{5}$$

We assume that each VM is charged by hour, and hence, if only a small fraction of the hour is utilised, then the user still has to pay for the entire hour. The cost of running a $vm \in VM$ is

$$cost_{vm} = \lceil \frac{exec_{vm}}{3600} \rceil \times c_{pt} \tag{6}$$

The overall time to complete all tasks is the execution time of the slowest VM (all VMs execute tasks in parallel) and is denoted as

$$exec = \max_{vm \in VM} exec_{vm} \tag{7}$$

The total cost to execute all tasks is the sum of the costs of all VMs which is

$$cost = \sum_{vm \in VM} cost_{vm} \tag{8}$$

If B denotes the budget constraint for the amount of money that can be spent for executing T on the cloud, then

$$cost \leq B \tag{9}$$

In this research, the performance of BoTs on the cloud is maximised by determining VM, referred to as an **execution plan**, which contains a set of VMs and the assignment of tasks onto the VMs, so that the overall execution time, $exec$, is minimised while satisfying the budget constraint.

IV. Heuristic Algorithm

This section presents the algorithm used to solve the problem of executing multiple BoT applications on the Cloud. The main steps of the algorithm include, creating VMs, assigning tasks to VMs, balancing tasks between VMs, generating an initial plan based on local performance, adding more VMs based on the user's budget, keeping VMs' execution times under one hour, replacing expensive VMs by cheaper ones and finding an execution plan.

Our approach to address the problem consists of algorithms which are presented in Sections IV-A to IV-G. Section IV-H presents the complete approach.

A. Assign Tasks To VMs

Function $ASSIGN$ aims to assigns a list of tasks to a given list of VMs. For each task, a receiving VM is selected based on three criteria: i) the cost of a VM should not increase if a task is executed in it, moreover, a receiving VM should ii) require the least time to execute a task and iii) has the lowest execution time in comparison to other VMs.

After the assignments there may be VMs without any assigned tasks, since their instance types do not have the best performance for any task.

B. Balance Tasks Between VMs

When tasks are assigned to VMs of different types, it is possible to have one VM with a higher execution time than the others. As shown by Equation 7, this will affect the overall execution time. Hence, it is necessary for tasks to be evenly distributed among all VMs so that their execution can be completed nearly at the same time. This process is performed by the function $BALANCE$ which moves tasks from VMs with highest execution times to other ones as long as the overall execution time does not increase.

C. Create Initial Plan by Selecting Instance Type with Best Performance for each Application

The best instance type of an application is the one whose cost is lower than the given budget and maximises performance of an application. If there are multiple instance types that maximise application performance, then the cheapest one is selected: $it^b_{A_i} = \arg\min_{it \in IT} (P_{it,A_i}, c_i t)$.

In the initial plan generated by function $INITIAL$, the tasks are assigned to the best instance type. In other words, an application's tasks are assigned to the number of VMs of the same instance type.

For each application, the whole budget is used to hire VMs of its best instance type: $num = \lfloor \frac{B}{it^b_{A_i}} \rfloor$. As there are many applications, the budget is likely to be violated.

D. Reduce cost

As an initial plan is highly likely to violate the budget constraint, the next step, therefore, is to reduce the overall cost until the budget constraint is satisfied.

Moving task can potentially increase the cost if it results in an additional hour added for the receiving VM. So, the goal of the cost reduction process is to **completely remove a number of VMs by moving all of their tasks to other VMs** without increasing the overall cost.

The cost reduction is performed using function $REDUCE$ which tries to move **all** tasks from one VM with lowest execution time to others. The function has two modes, **local mode** only allows tasks to be moved to VMs of the same type of an initial VM, while **global mode** allows tasks to be moved to VM of any type. In order to keep task's execution time as low as possible, the function tries to move tasks to VMs whose require least time to execute them.

E. Add More VMs based on Budget

Until this stage, only the best performing VMs are used. Based on the **remaining budget**, a few more VMs can be added to increase the execution concurrency which results in lower execution time even though they are not best performing.

Function ADD aims to add the most number of VMs based on the remaining budget $B_r = B - cost$. The instance type of the added VMs is the cheapest one with the lowest execution time for all tasks. By assuming that each of them would not be executed for more than one hour, it is possible to calculate a cost for a new VM, and the function keeps added new VMs until there is not enough money to add any more.

F. Keep VM's Execution in One Hour

As cloud VMs are usually charged by the hour; running a VM for two hours will be similar in cost to running two VMs of the same type in parallel for one hour. Hence, we introduce function $SPLIT$ which keeps assigning tasks from a VM whose execution time is more than one hour to two VMs with the same instance type as long as the budget constraint is not violated and overall execution time decreases.

G. Replace Expensive VMs by Cheaper Ones

Sometimes, it is cost-effective to use a large numbers of cheaper and moderately performing VMs than fewer expensive and high-performing VMs. For example, assuming there are two instance types $IT = \{it_1, it_2\}$ and one application with 10 tasks of size 1: $A = \{A_1\}$. The cost and performance of it_1 are \$2 and \$8, which means a VM of instance it_1 costs \$2 per hour and takes 8 seconds to execute one task of A_1. Similarly, the cost and performance of it_2 are \$1 and \$10. With the budget $B = \$2$, it is possible to have one VM of type it_1 and takes $8 \times 10 = 80$ seconds to execute all ten tasks of A_1. Alternatively, with the same budget, two VMs of type it_2 can be deployed. As tasks are evenly distributed to both VMs, each VM executes five tasks and takes $10 \times 5 = 50$ seconds to complete execution. The execution when two VMs of instance it_2 are employed is 50 seconds. In this case, two VMs of type it_2 perform better.

Function $REPLACE$ aims to replace expensive VMs with cheaper ones in order to increase the cost-effectiveness of the execution. First of all, it selects the certain number of VMs and find their cost. Then, it calculates how many VMs of the **cheaper** instance type are affordable based on the cost and the remaining budget (if there is any). For simplification, only one instance type is considered of the time, which means the set of VMs has the same instance type. All tasks from the selected VMs are assigned to the set of new and cheaper VMs. After assignment, if the budget is still satisfied and the overall execution time is reduced, the selected VMs are officially replaced.

H. Find an Execution Plan based on the Given Budget Constraint

Algorithm 1 is used to find the execution plan based on the given budget constraint using all functions introduces in the previous sections. First of all, the $INITIAL$ function is called to create an initial plan, in which all tasks are assigned to VMs of their best instance types possible, which are then locally reduced (Lines 2, 3 and 4).

For future comparison, the current plan, cost and execution time are stored (Lines 7, 5 and 6).

After that, the current plan is globally reduced, in which tasks can be moved to all VMs except the one which is selected to be removed. (Line 9). Additional VMs can be added if it is allowed by the remaining budget (Line 10) and tasks are balanced between all VMs (Line 11). Then, we try to keep the execution to all VMs under one hour (Line 12). As it is not guaranteed that the current execution plan satisfies the budget constrain, the greater value between the real one and the current cost of the execution is used as an temporary budget for $REPLACE$ function, which tries to replace expensive VMs which more cheaper ones (Line 13).

The Algorithm is an iterative process which tries to optimise the execution plan by reducing its cost and execution time. Hence, if the current plan is better than the previous one, i.e. the execution time of the cost are reduced, the iteration continues (Line 14). Otherwise, if there is no improvement in term of cost and execution time, the plan is returned (Line 19).

Algorithm 1 Find

```
1:  function DO_ASSIGNMENT(T, IT, B)
2:      VM ← INITIAL(A_T, IT, B)
3:      VM ← ASSIGN(T, VM)
4:      VM ← REDUCE(VM', B, ∅, TRUE)
5:      cost' ← MAX_NUMBER
6:      exec' ← MAX_NUMBER
7:      VM' ← VM
8:      loop
9:          VM ← REDUCE(VM', B, ∅, FALSE)
10:         VM ← ADD(IT, VM, B − cost)
11:         VM ← BALANCE(VM)
12:         VM ← KEEP(VM)
13:         VM ← REPLACE(IT, VM, max B, cost, 1)
14:         if cost < cost' ∨ exec < exec' then
15:             cost' ← cost
16:             exec' ← exec
17:             VM' ← VM
18:         else
19:             return VM'
20:         end if
21:     end loop
22: end function
```

V. Evaluation

This section evaluates our approach by comparing its performance with two approaches.

A. Approaches for Comparison

The approaches used for comparing our algorithm are as follows:

1) Minimising Individual Task Execution Time (MI) Approach: this approach aims to minimise the execution time of any individual task by selecting the instance type which has the best performance among all tasks. It can be easily performed by invoking Algorithm ADD with full budget.

2) Maximising Parallelism (MP) Approach: in this approach, the cheapest instance type is selected so that the maximum number of VMs can be purchased based on the given budget $it^c = \arg\min_{it \in IT}(c_{it})$.

B. Environment Setup

We built a simulation framework using Scala to evaluate the heuristic algorithm. The framework models multiple instance types of a cloud with different performance and varying costs as shown in Table I; this is input to the simulation. In cloud environment, those inputs can be obtained by sampling the applications, i.e. running the small amount of their tasks, on VMs of difference instance types. The framework then uses Algorithm 1 to generate an execution plan. This plan is then executed for obtaining the overall cost and time.

1) Applications: Three application A_1, A_2, A_3 were considered in the experiments. The first one used the same amount of compute and memory resources and the other two were CPU and memory intensive applications. Each application consisted of 250 tasks whose side are equally distributed from 1 to 5.

2) Instance Types: We assumed that there were four instances types it_1, it_2, it_3, it_4. The first one was very cheap and had poor performance for all applications. The second one was a general instance type which provided the balance between compute and memory. The third and forth ones were compute and memory optimised instance types which were most suitable for CPU and memory intensive applications, respectively. The last three instance types had the same cost which was twice in comparison to the first one.

3) Cost and Performance: The description, cost and performance of each instance type is presented in Table I. It can be seen that even though the last three instance types had the same price, they performed differently.

Instance Name	Description	Cost	Performance		
			A_1	A_2	A_3
it_1	Small general type	5	20	24	22
it_2	Big general type	10	11	13	12
it_3	CPU optimised type	10	10	15	9
it_4	Memory optimised type	10	10	9	12

TABLE I: Costs and Performances

4) Budget: The budget constraint was set to different values ranging from 40 to 85.

C. Experimental Results

The result is shown in Figure 1. The x-axis represents the budget while the y-axis is the execution time. The black horizontal dashed line represents 3600 seconds, i.e. an hour.

It can be seen that, given the same budget, our approach, i.e. red and triangle line, always had the lower execution time in compare to other 2 simple approaches. In comparison with the MI approach, ours was able to reduce the execution time by average 13%. The MP approach which focused on maximising the execution parallelism by choosing the cheapest instance type performed better than MI approach, which preferred more expensive instance type. However, in average, its execution times were still 7% higher than the proposed one.

Furthermore, our approach was also able to handle the low budget constraint: while MP required the budget to be at least 45 and MI could not satisfy any budget below 50, our approach satisfies the budget as low as 40.

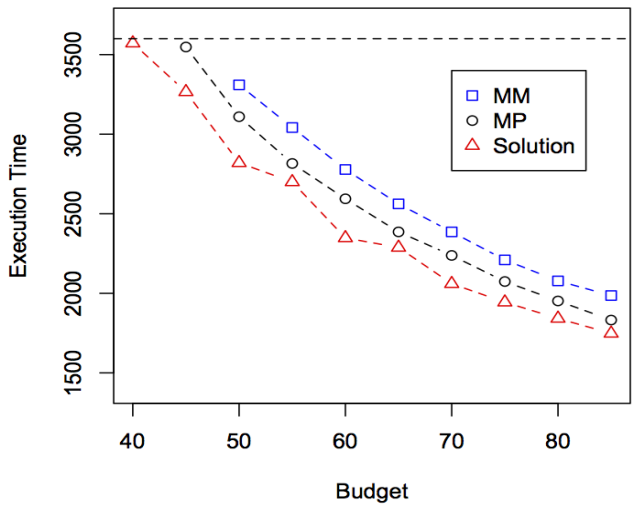

Fig. 1: Execution Times for Different Approaches

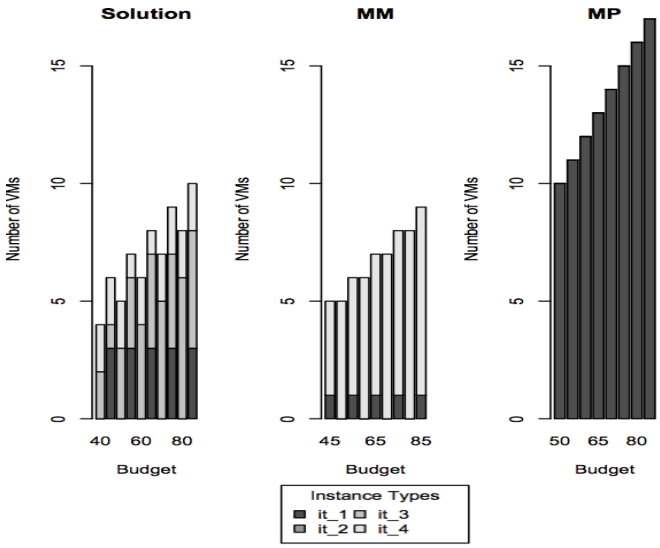

Fig. 2: Number of VMs of Each Type

As mentioned earlier, there is a trade-off between minimising an individual task's execution time with maximising the parallelism. The trade-off is presented based on instance selection: powerful but expensive versus less powerful but cheap instance types. Moreover, using the combination of different instance types usually results in better performance in compare to selecting one instance type.

Figure 2 shows the number of VMs and their instance types used by different approaches for different values of the budget constraint. It can be seen that the MP (right figure) approach always went for cheapest instance type (i.e. it_1) and managed to maintain the highest number of VMs. On the other hand, the MI approach (middle figure) tried to use as much VMs of instance type it_4 as possible since it had the best average performance in compare to other three. If there were any remaining budget, MI added an additional VM of it_1 in order

to increase the performance.

Instead of following only on trend to select instance type, our approach (left figure) was more flexible. When the budget is 40, 50, 60, 70 and 80, it prioritised execution parallelism by adding more VM of the cheapest instance type it_1. However, then the budget is 45, 55, 65, 77 and 85, none VM of it_1 was created. Instead, VMs of it_3 and it_4, which had the best performance for tasks of A_2 and A_3, were created in order to reduce the overall individual task execution time. As the result, our approach could achieve the better performance with the same budget constraint.

VI. Conclusion

BoT applications have been widely used in not only scientific but also industrial communities. However, they require a huge amount of resources which can only be satisfied in a distributed environment consisting of many interconnected machines. Many efforts have been spent on optimising the execution of BoT applications on grid computing in which resources are already available and users have to compete with each other to acquire free resources. Hence, the scheduling of BoT on the Grid mainly focuses on assigning tasks to the 'best' suited machines.

Cloud computing on the other hand provides an isolated environment (not taking into account multitenancy) in which a user does not need to share her resources with anyone else. Moreover, it is also possible to select the resource types which are best suited for the applications. However, cloud computing resources are not free of charge and a user has to pay as soon as the VMs start running. Hence, the problem of executing BoT applications on the cloud is not only about assigning tasks to resources but also selecting the type of resource(s), which are most appropriate. Moreover, with multiple applications to run, the problem is further complicated as each application potentially requires different types of resource for to achieve the best performance.

In this paper, we investigated the execution of multiple BoT applications on the cloud given a budget constraint. The problem is modelled and a heuristic algorithm was proposed in order to decide the selection of different cloud resources and the assignment of tasks onto resources. By comparing our approach to other simple ones, it was shown that the proposed heuristic algorithm was able to reduce the execution time from 4% to 15% given the budget constraint.

For future work, we plan to further expand our heuristic algorithm to take into account the execution deadline while minimising the cost. Moreover, we also want to incorporate dynamic scheduling feature to handle any unexpected issues during runtime, which are inevitable in real-time execution on the Cloud. Finally, we want to support scheduling tasks whose execution times are unknown, i.e. non-clairvoyant scheduling.

Acknowledgment

This research is supported by the EPSRC grant 'Working Together: Constraint Programming and Cloud Computing' (EP/K015745/1), a Royal Society Industry Fellowship 'Bringing Science to the Cloud', an EPSRC Impact Acceleration Grant (IAA) and an Amazon Web Services (AWS) Education Research Grant.

References

[1] A. Iosup and D. Epema, "Grid computing workloads," *Internet Computing, IEEE*, vol. 15, pp. 19–26, March 2011.

[2] A. Barker, B. Varghese, J. S. Ward, and I. Sommerville, "Academic cloud computing research: Five pitfalls and five opportunities," in *6th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud '14*, 2014.

[3] "Boinc." http://boinc.berkeley.edu/. Accessed: 2014-01-23.

[4] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. Silva, C. Barros, and C. Silveira, "Running bag-of-tasks applications on computational grids: the mygrid approach," in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pp. 407–416, Oct 2003.

[5] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the Eighth Heterogeneous Computing Workshop*, HCW '99, pp. 30–, IEEE Computer Society, 1999.

[6] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, HPDC '02, (Washington, DC, USA), pp. 352–, IEEE Computer Society, 2002.

[7] C. Weng and X. Lu, "Heuristic scheduling for bag-of-tasks applications in combination with qos in the computational grid," *Future Gener. Comput. Syst.*, vol. 21, pp. 271–280, Feb. 2005.

[8] K. Kaya and C. Aykanat, "Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, pp. 883–896, Aug 2006.

[9] S. Venugopal and R. Buyya, "A deadline and budget constrained scheduling algorithm for escience applications on data grids," in *in Proc. of 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005*, pp. 60–72, Springer-Verlag, 2005.

[10] R. Bertin, A. Legrand, and C. Touati, "Toward a fully decentralized algorithm for multiple bag-of-tasks application scheduling on grids," in *Grid Computing, 2008 9th IEEE/ACM International Conference on*, pp. 118–125, Sept 2008.

[11] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien, "Scheduling concurrent bag-of-tasks applications on heterogeneous platforms," *Computers, IEEE Transactions on*, vol. 59, pp. 202–217, Feb 2010.

[12] C. Anglano and M. Canonico, "Scheduling algorithms for multiple bag-of-task applications on desktop grids: A knowledge-free approach," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, April 2008.

[13] A. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 351–359, Nov 2010.

[14] M. H. Farahabady, Y. C. Lee, and A. Y. Zomaya, "Non-clairvoyant assignment of bag-of-tasks applications across multiple clouds," in *Proceedings of the 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '12, (Washington, DC, USA), pp. 423–428, IEEE Computer Society, 2012.

[15] J. O. Gutierrez-Garcia and K. M. Sim, "A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling," *Future Gener. Comput. Syst.*, vol. 29, pp. 1682–1699, Sept. 2013.

[16] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pp. 41–48, Oct 2010.

[17] L. Thai, B. Varghese, and A. Barker, "Executing bag of distributed tasks on the cloud: Investigating the trade-offs between performance and cost," in *6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014)*, 2014.

[18] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for hadoop: Time to rethink?," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, (New York, NY, USA), pp. 20:1–20:13, ACM, 2013.