A Performance-oriented Resource Allocation Algorithm with Insertion and Duplication for IaaS Clouds

Inseong Song, Jinhyuk Kim, Sangbang Choi

Department of Electronic Engineering, Inha University, Korea
nicthevirus@gmail.com, graybaily@inha.edu, sangbang@inha.ac.kr

Abstract

The innate characteristics of a cloud computing environment make itself attractive for running applications with large scale data, as a user can utilize a number of high performance computing virtual machine instances without maintenance cost. The performance of executing an application in a cloud computing environment depends on a resource allocation policy that schedules an input task graph onto virtual machine instances. In this paper, we propose a novel resource allocation policy that considers characteristics of an input task graph and virtual machine instances of a cloud computing environment. And also, we propose a new task insertion method that also considers an execution finish time of a child task of the inserted task. Simulation experiments with task graphs from standard task graph project show that the proposed algorithm outperforms conventional algorithms in terms of normalized total execution time.

Keywords: Cloud computing, DAG, Scheduling, Task insertion, Task duplication

1 Introduction

The demands for rapid processing of applications with large scale data increase rapidly in fields of science and technology, such as weather forecast, fluid dynamics, and space programs. Along with the growing needs for processing large scale data, a cloud computing environment, especially the Infrastructure as a Service (IaaS) clouds, gets attention with its innate characteristics; its ability to execute a numerous number of VM (Virtual Machine) instances at the same time, and to provide on-demand computing resource provisioning without maintenance cost [1-6]. Over the last few years, the IaaS clouds have grown and matured rapidly. Leading cloud providers, such as Amazon and Google, are diversifying their instance types and services including instances for high performance computing. Amazon EC2 provides 13 types of type C VM instances, and Google Compute Engine provides 16 types of type n VM instances for high performance computing [7-8]. For example, Amazon EC2 c4.2xlarge VM instance offers 8 virtual CPUs and 15GB memory, and Google Compute Engine n1-standard-8 VM instance offers 8 virtual CPUs and 30GB memory.

A large scale application that is to be executed on a cloud computing environment can be modeled as a Directed Acyclic Graph (DAG). And the total execution time of an application depends on a method that schedules tasks in an input DAG onto a target cloud computing environment [9]. However, the task scheduling problem is an NP-complete problem which cannot be finished in a polynomial time. And as a result, most of the previous researches concentrate on obtaining sub-optimal performance-effective solutions [10-13], or minimizing user cost within the user specified time limit [4, 14-16] with scheduling heuristics.

Common scheduling heuristics consist of two phases, a task prioritizing phase and a resource allocation phase. The task prioritizing phase calculates and decides a priority of each task in an input DAG, and the resource allocation phase allocates those tasks on cloud computing resources according to a resource allocation policy. The resource allocation policy is a scheme that makes a task to be allocated on a case optimal cloud computing resource. There are two advanced resource allocation policies which try to enhance the performance in terms of execution time; an insertion policy that utilizes a task insertion method and a duplication policy that utilizes a task duplication method [9].

In this paper, we propose a novel resource allocation policy called an Advanced Hybrid Allocation Policy (AHAP), and a new task insertion method of which an AHAP utilizes. The AHAP uses both an insertion method and a duplication method according to the characteristics of a target cloud computing environment and an input DAG, while conventional policies use either one of them, or none of them. The AHAP compares the computation cost and the communication cost of an input DAG to the heterogeneity of cloud computing resources. And then, allocates a task on an optimal resource with either a new insertion method proposed in this paper or a duplication method, after

*Corresponding Author: Sangbang Choi; E-mail: sangbang@inha.ac.kr
DOI: 10.3966/160792642018011901021

comparing the values with a proposed criteria. The new insertion method proposed in this paper tries to reduce an execution finish time of a child task of the inserted task, not just reducing an execution finish time of an inserted task like conventional one does.

We have designed and implemented a simulator to evaluate the performance of the proposed resource allocation policy. For an unbiased simulation, we have used task graphs in STanDard task Graph Project (STDGP) [17] and task graphs that model real world applications, such as Fast Fourier Transform (FFT) and Gaussian Elimination (GE). Simulations are carried out by applying the AHAP to conventional task scheduling algorithms which aim to achieve the best performance in terms of execution time, such as Heterogeneous Critical Parents with Fast Duplicator (HCPFD) algorithm [10], Heterogeneous Critical Task (HCT) [11], and Performance Effective Task Scheduling (PETS) algorithm [12]. And the combined algorithms, the HCPFD with the AHAP, the HCT with the AHAP, and the PETS with the AHAP, are compared to its original algorithms, the HCPFD, the HCT, and the PETS, respectively. The main metric of the performance comparison is normalized total execution time, the total execution time divided by the shortest execution time of an input DAG. Comparison results show that the AHAP shows up to 7.74%, 5.48%, and 7.52% better performance than the HCPFD, the HCT, and the PETS in terms of normalized total execution time with respect to the number of tasks in a graph, respectively.

The rest of this paper is organized as follows. In section 2, we present the task scheduling problem. In section 3, brief descriptions about conventional task scheduling algorithms are presented. Section 4 describes the proposed AHAP algorithm and improved insertion method, and the performance evaluation results of the proposed algorithm are presented in section 5. Section 6 summarizes and concludes this paper.

2 Problem Definition

Input task graphs, which model applications with large scale data that are to be executed on a cloud computing environment, are generally modeled as a DAG $G=(V, E)$. V is a set of tasks which consist an application, and E is a set of edges which represent precedence constraints [9]. Figure 1 shows an example of a DAG. A number on an edge stands for a weight, which is an average communication cost between two tasks.

A DAG to be executed has to have one entry task and one exit task. In Figure 1, task 1 and task 10 are entry task and exit task, respectively. If a DAG has multiple entry tasks or exit tasks, a dummy task with no computation cost and communication cost is added to the DAG.

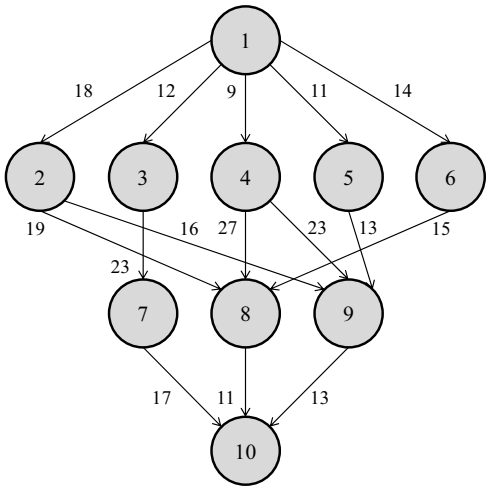

Figure 1. An example of a DAG

A task set $V=\{v_1, v_2, ..., v_n\}$ includes all tasks in a DAG, where an ith task in the set is expressed as v_i. A task set with the number of n tasks means that an application consists of n tasks. An edge set $E=\{e_{1,2}, e_{1,3}, ..., e_{i,j}\}$ includes all edges in a DAG, which connect tasks in a DAG. An edge connects two tasks with direction, and thus can express precedence constraint. In other words, an edge $e_{i,j}$ connects two tasks v_i and v_j, and this indicates that v_j can start its execution after v_i finishes its execution. In this case, v_i is called as a parent task of v_j, and v_j is called as a child task of v_i.

A child task set $succ(v_i)$ includes all child tasks of a task v_i. Every task in a DAG has at least one child task, except an exit task. A parent task set $pred(v_i)$ includes all parent tasks of v_i. Every task in a DAG has at least one parent task, except an entry task. A critical parent task, expressed as $dpred(v_i)$, is a task that passes data to v_i on the latest time.

A cloud computing environment provides computation service set $S=\{S_1, S_2, ..., S_n\}$ as form of VMs. Each service type, or a VM type, offers different computation power. $S_{i,j}$ is a provisioned ith instance of a VM type S_j.

As a DAG is to be processed on a cloud computing environment which offers different computation power for each VM type, an estimated computation cost of a task on each VM type is given as a table W. And a computation cost of a task v_i on an instance $S_{j,k}$ is notated as $w(v_i, S_{j,k})$. Table 1 shows an example computation cost of a DAG on Figure 1, on 3 VM types. As shown in the Table 1, each VM type has given a different estimated computation cost as a cloud computing environment is heterogeneous.

An average communication cost $\bar{c}(v_i, v_j)$ is the time elapsed while transferring data from a task v_i to a task v_j over a network. If v_i and v_j are executed on a same instance, an average communication cost becomes 0. Although there is still a need to transfer data between tasks, the cost is negligible since it is a local job. An average computation cost $\bar{w}(v_i)$ is an

Table 1. An example computation cost table

Task	S_1	S_2	S_3
1	14	16	9
2	19	18	13
3	19	13	11
4	17	13	8
5	13	12	10
6	16	13	9
7	15	11	7
8	14	11	5
9	20	18	6
10	21	16	7

average computation cost of a task v_i on every provisioned instances, and is calculated with equation (1).

$$\overline{w}(v_i) = \sum_{all\ provisioned\ instances} \frac{w(v_i, S_{j,k})}{n} \quad (1)$$

where n is the number of all provisioned instances.

An Instance Available Time $IAT(S_{i,j})$ is the time that the instance $S_{i,j}$ completes the execution of a previous task and is ready to execute another task. If a task v_k is the last allocated task on a $S_{i,j}$, then the execution finish time of the v_k is $IAT(S_{i,j})$.

An Earliest Start Time $EST(v_i, S_{k,l})$ is the earliest execution start time of a task v_i on an instance $S_{k,l}$, and is expressed with equation (2) [13]. An Earliest Finish Time $EFT(v_i, S_{k,l})$ is the earliest execution finish time of a task v_i on an instance $S_{k,l}$. It is the sum of $EST(v_i, S_{k,l})$ and a computation cost of v_i on $S_{k,l}$, and is calculated with equation (3).

$$EST(v_i, S_{k,l}) = \begin{cases} 0 & v_i = v_{entry} \\ max_{v_j \in pred(v_i)} \{IAT(S_{k,l}), max\{EFT(v_j) + \overline{c}(v_i, v_j)\}\} & otherwise \end{cases} \quad (2)$$

$$EFT(v_i, S_{k,l}) = EST(v_i, S_{k,l}) + w(v_i, S_{k,l}) \quad (3)$$

We assume followings in this paper. A cloud computing environment provides enough communication bandwidth so that the multiple communications can take place at the same time without delays. Also, an instance can compute a task, and send or receive data simultaneously. Execution of a task cannot be preempted or checkpointed. And, the overheads related to the task scheduling are ignored.

3 Related Work

As stated above, scheduling a DAG on a cloud computing environment is an NP-complete problem. Therefore, solving the problem for the best solution in polynomial time is nearly impossible. Instead of finding the best solution, researches on finding sub-optimal solutions have been briskly carried out. In this chapter, we briefly describe well-known simple and effective task scheduling algorithms that are used in a heterogeneous high performance computing environment and a cloud computing environment. The algorithms aim to achieve superior performance in terms of execution time or minimize user cost required for leasing a computing resource.

3.1 Algorithms for Reducing Execution Time

PETS. The PETS algorithm [12] aims to achieve the best performance with low complexity. The algorithm utilizes a graph leveling method and an insertion method. The algorithm consists of three phases, leveling phase, task prioritizing phase, and resource allocation phase. In leveling phase, tasks in each level, i.e. tasks that are independent to each other, are grouped together. In task prioritizing phase, a priority value of each task is calculated according to a specific priority function, called a $rank(v_i)$. A $rank(v_i)$ is a sum of computation costs and communication costs of tasks and edges on a critical path, from an entry task to a task v_i. After calculating $rank(v_i)$ values, an ordered priority list is generated by sorting the tasks in increasing order of rank value. If there are multiple tasks with the same $rank(v_i)$ value, a task with smaller computation cost gets priority. In resource allocation phase, the algorithm calculates execution finish time of a task on each computation resource considering possible insertions, and allocate the task on the resource which provides the earliest finish time of the task according to the order decided in task prioritizing phase. The PETS algorithm provides good schedule with low complexity, but has relatively low performance than algorithms with a duplication method.

HCPFD. The HCPFD algorithm [10] is a well-known duplication based scheduling algorithm. The algorithm aims to achieve the best performance with all possible computing resources. The algorithm consists of two phases, task prioritizing phase, and resource allocation phase. In task prioritizing phase, the algorithm traverses an input DAG upward, and computes an average latest execution start time and an average earliest execution start time of each task in the DAG. Then, set the tasks with 0 differences between two values as critical tasks, which construct a critical path. And for each critical task from an exit task, the algorithm puts parent tasks of the critical task to the priority list followed by the critical task. In resource allocation phase, the algorithm checks if duplicating a critical parent of a task at the idle time between already scheduled tasks on a computing resource is possible, and if duplicating the critical parent of the task reduces execution finish time of the task. If these two conditions are met, the algorithm duplicates the critical parent of the task, and allocates the task on the same

resource where the parent is allocated on. If the conditions do not met, the algorithm allocates the task on the resource which provides the fastest execution finish time. The HCPFD algorithm provides superior schedule, but has high complexity when compared to algorithms with the other allocation policies and can have redundant duplications.

HCT. The HCT algorithm [11] is a duplication based scheduling algorithm which aims to achieve the best performance. The algorithm consists of two phases, scheduling list construction phase, and task assignment phase. In scheduling list construction phase, a priority value of each task is calculated according to a priority function, called an *upward rank*. An *upward rank* of a task is a sum of computation costs and communication costs of tasks and edges on a critical path, from an exit task to the task. Then, the scheduling list is constructed by sorting the tasks in decreasing order of their *upward rank*. In task assignment phase, the algorithm checks if duplicating a critical parent of a task reduces execution start time of the task on each computing resource. If the condition is met, the algorithm duplicates the critical parent of the task, and allocates the task on the same resource where the parent is allocated on. If the condition do not met, the algorithm allocates the task on the resource which provides the fastest execution finish time. But despite utilizing a duplication method, the HCT algorithm does not provide good schedules when compared to other algorithms for reducing execution time.

3.2 Algorithms for Minimizing User Cost

IC-PCP. The IaaS Cloud – Partial Critical Path (IC-PCP) algorithm [14] is an up-to-date scheduling algorithm which takes account of a user cost. Unlike previous algorithms, the IC-PCP algorithm tries to minimize cost required for leasing a computing resource of a cloud computing environment while satisfying the user specified time limit. The algorithm consists of two phases, partial critical path generation phase, and resource allocation phase. In partial critical path generation phase, the algorithm builds a critical path from an exit task with the tasks that are not assigned yet, a partial critical path. And with the partial critical path, the algorithm runs to the resource allocation phase. In resource allocation phase, the algorithm allocates the partial critical path to the cheapest computation resource while not violating the time limit. If there is no place to allocate it on currently leased resources, the algorithm newly leases a resource and allocates it. And then, the algorithm goes backwards to critical path generation phase, and creates a new partial critical path. However, the IC-PCP algorithm only concentrates on minimizing cost and as a result, the algorithm is not capable of providing good performance.

POSH. The Pareto Optimal Scheduling Heuristic (POSH) algorithm [15] is a multi-objective scheduling algorithm. The algorithm aims to solve the optimization problem between reducing execution time and minimizing cost. The algorithm uses Pareto efficiency to solve the problem, and find the Pareto optimal solution between execution time and cost. The algorithm consists of two phases, prioritizing phase, and mapping phase. In prioritizing phase, a sorted list is created in decreasing order of priority. A priority value of a task is a sum of computation costs and communication costs of tasks and edges on a critical path, from an exit task to the task. In mapping phase, the algorithm assigns tasks in a sorted list to resources based on Pareto dominance, which is calculated from an objective function that the authors have proposed. The POSH algorithm can offer an affordable execution time with an affordable cost. However, the algorithm cannot provide superior performance in terms of both execution time and cost.

4 Proposed Algorithm

In this paper, we propose a novel resource allocation policy for task scheduling algorithms, the AHAP algorithm. Previous resource allocation policies allocate a task onto resources without considering the characteristics of a cloud computing environment and an input DAG, and use either an insertion method or a duplication method, or none of them. The AHAP, however, uses both an insertion method and a duplication method after deciding which resource allocation method should be used according to an average communication to computation ratio of an input DAG and a heterogeneity of provisioned cloud computing resources. In addition, the AHAP offers a new insertion method that allocates tasks more effectively considering a finish time of a child task.

4.1 Task Insertion and Task Duplication

Lots of data transfers may occur while executing an application with large scale data, since tasks in a DAG have lots of precedence constraints. And because a child task cannot start its execution before receiving data from a parent task, a finish time of an application can be delayed. There are two ways to reduce the delay occurred from transferring data; a task insertion method and a task duplication method. Both provide superior scheduling results when compared to ones not using them.

Figure 2 and Figure 3 shows brief examples of a task insertion and a task duplication. In Figure 2 and Figure 3, a task 4, a non-shaded one, has to receive data from a task 2 before its execution, and has no precedence constraints with a task 3. In case of task insertion in Figure 2, if there is enough space for the task 4 on between already scheduled task 1 and task 3, if the insertion reduces the execution finish time of the task 4, and if the insertion does not violates the precedence

constraints, the insertion of the task 4 is carried out in between task 1 and task 3. As a result, the task 4 is moved from the location before to the location after [13]. In case of task duplication in Figure 3, if a duplication of the task 2 reduces the execution finish time of the task 4, the duplication of the task 2 is carried out and the task 4 is moved from the location before to the location after [10].

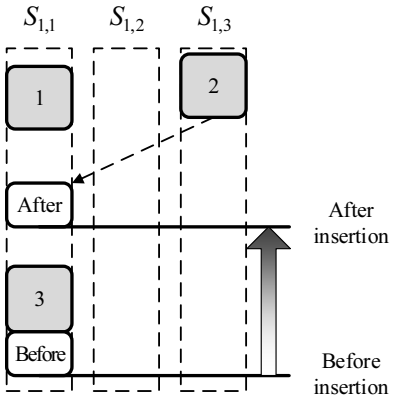

Figure 2. An example of a task insertion

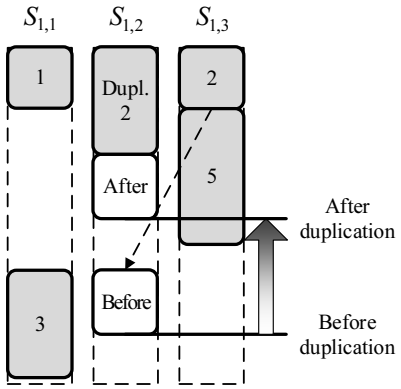

Figure 3. An example of a task duplication

4.2 Proposed AHAP Algorithm

As stated above, lots of data transfers occur while executing an application with large scale data, and they delay the execution of the application. And there are two well-known methods which reduce the delay occurred from transferring data; an insertion method and a duplication method. To reduce the delays, the AHAP algorithm utilizes both an insertion method and a duplication method, according to the characteristics of an input DAG and a target cloud computing system.

The AHAP algorithm assumes that all computing resources are already provisioned as a form of VM instances. In a high level view, the algorithm consists of three steps; communication to computation cost ratio calculation step, heterogeneity factor calculation step, and allocation step.

In a first step, the AHAP algorithm calculates a communication to computation ratio *ccr* of an input DAG [9]. To calculate the ratio, an average communication cost of all edges in an input DAG is calculated, followed by calculating an average computation cost of all tasks in an input DAG. Then, the average communication cost is divided into the average computation cost. The result is a communication to computation ratio *ccr*.

In a second step, for each task in an ordered task list, the AHAP algorithm calculates a heterogeneity factor $\beta(v_i)$. A heterogeneity factor stands for the difference of computation cost among the instances. The algorithm calculates a heterogeneity factor $\beta(v_i)$ of instances for a task v_i with the following equation (4).

$$\overline{w}(v_i)\times\left(1-\frac{\beta(v_i)}{2}\right)\le w(v_i,S_{j,k})\le\overline{w}(v_i)\times\left(1+\frac{\beta(v_i)}{2}\right) \quad \textbf{(4)}$$

The smallest $\beta(v_i)$ value which satisfies the equation (4) for all provisioned instances $S_{j,k}$ is a heterogeneity factor of instances for v_i.

In a third step, the AHAP algorithm decides which allocation method has to be used. Before deciding an allocation method, a criteria for deciding an allocation method has to be determined. According to equation (4), the possible maximum computation cost of a task v_i can be expressed as in the following equation (5).

$$w_{\max}(v_i,S_{j,k})=\overline{w}(v_i)\times\left(1+\frac{\beta(v_i)}{2}\right) \quad \textbf{(5)}$$

And according to the definition of a *ccr*, an estimated communication cost from a task v_i to its child task v_j, $c_{estimated}(v_i,v_j)$, is expressed as equation (6).

$$c_{estimated}(v_i,v_j)=ccr\times\overline{w}(v_i) \quad \textbf{(6)}$$

Next, to determine a comparison criteria, we compare the estimated communication cost with the possible maximum computation cost with the equation (7).

$$c_{estimated}(v_i,v_j)>\overline{w}(v_i)\times\left(1+\frac{\beta(v_i)}{2}\right) \quad \textbf{(7)}$$

By combining the equation (6) and the equation (7), we can obtain the equation (8).

$$ccr\times\overline{w}(v_i)>\overline{w}(v_i)\times\left(1+\frac{\beta(v_i)}{2}\right) \quad \textbf{(8)}$$

And finally, we are able to obtain the final equation (9) by eliminating $\overline{w}(v_i)$ on both sides.

$$ccr>1+\frac{\beta(v_i)}{2} \quad \textbf{(9)}$$

The AHAP algorithm utilizes the equation (9) as a criteria to decide which allocation method has to be used. For each task v_i in an ordered task list, except an entry task, find $dpred(v_i)$ and apply the equation (9) to the $dpred(v_i)$.

If the equation (9) is satisfied, the AHAP algorithm

judges that there might be enough idle time slots on instances to duplicate the critical parent task since the estimated average communication cost is bigger than the possible maximum computation cost of the critical parent task, and tries to duplicate the critical parent task. On the other hand, if (9) is not satisfied, the algorithm judges that there is not enough idle time slots to duplicate the critical parent task, and only tries to insert the current task with a proposed improved insertion method. Following Algorithm 1 shows a pseudo code of the AHAP algorithm.

Algorithm 1. The AHAP algorithm

▷ First step
  Take an ordered task list L
  for each task in a DAG **do**
    Compute an average computation cost $\bar{w}(v_i)$
  endfor
  Calculate an average communication cost of a DAG
  Calculate an average computation cost of a DAG
  Calculate a communication to computation ratio *ccr*
▷ Second step
  for each task in L **do**
    Calculate a heterogeneity factor β
  endfor
▷ Third step
  Allocate an entry task v_{entry} on an instance $S_{j,k}$ that provides $EFT(v_{entry}, S_{j,k})$
  Remove v_{entry} from L
  for the first task v_i in L **do**
    Find $dpred(v_i)$
    if $ccr > 1 + \beta(dpred(v_i)) / 2$
      Allocate v_i with a *duplication_method*(v_i, L)
    else
      Allocate v_i with an *improved_insertion_method*(v_i, L)
    endif
  endfor

If the AHAP algorithm decides to use a duplication method, it first calculates *EFT*s of the current task on each provisioned instances and selects an instance $S_{j,k}$ that provides the minimum $EFT(v_i, S_{j,k})$. Then checks if there is enough time slots to duplicate the critical parent task $dpred(v_i)$ on the $S_{j,k}$, where the task v_i is scheduled. And also, assume that the $dpred(v_i)$ is duplicated on the $S_{j,k}$ and checks if the duplication of the $dpred(v_i)$ reduces the $EFT(v_i, S_{j,k})$. If the conditions are met, the $dpred(v_i)$ is duplicated on the $S_{j,k}$ followed by allocating the v_i on the $S_{j,k}$. If not, the v_i is allocated on the $S_{j,k}$ that provides the minimum $EFT(v_i, S_{j,k})$. Finally, the v_i is removed from L. Algorithm 2 shows a pseudo code of the duplication method.

Algorithm 2. A *duplication_method*(v_i, L) procedure

  Select an instance $S_{j,k}$ that minimizes $EFT(v_i, S_{j,k})$
  if enough time slots on $S_{j,k}$ to duplicate $dpred(v_i)$ and duplication of $dpred(v_i)$ reduces $EFT(v_i, S_{j,k})$ **do**
    Duplicate $dpred(v_i)$ on $S_{j,k}$
    Allocate v_i on $S_{j,k}$
  else
    Allocate v_i on $S_{j,k}$
  endif
    Remove v_i from L

If the AHAP algorithm decides to use an insertion method, especially an improved insertion method, it checks if following conditions are met.

Condition 1. For a task in an ordered task list, a next task in the list is a child task of the current task.

Condition 2. Under condition 1, the current task is a critical parent task of the next task, the child task.

Condition 3. Under condition 1, the current task and the next task are to be allocated on different instances.

Condition 4. Under condition 1 to 3, moving the current task to the instance where the next task is allocated reduces the execution finish time of the next task.

If all conditions are met, the AHAP algorithm allocates the current task where the next task is to be allocated to reduce the execution finish time of the child task, which can lead to a better schedule. If not, the AHAP algorithm uses a conventional insertion method to allocate the current task.

In improved insertion method, the AHAP algorithm first checks if a next task v_l in the ordered task list L is a child task of the current task v_i. If not, the v_i is allocated on an instance $S_{j,k}$ which provides the minimum $EFT(v_i, S_{j,k})$ with an insertion method, and the v_i is removed from L. If the v_l is a child task of the v_i, find an instance $S_{j,k}$ that minimizes $EFT(v_i, S_{j,k})$ with an insertion method and assume allocating the v_i on the $S_{j,k}$. With an assumption, find an instance $S_{m,n}$ that minimizes $EFT(v_l, S_{m,n})$ with an insertion method, and mark the execution finish time as an *EFT1*. Then, the AHAP algorithm checks if the v_i is a critical parent task of the v_l and if the $S_{j,k}$ does not equals to the $S_{m,n}$. If any of both does not meet, the v_i is allocated on the $S_{j,k}$ and the v_l is allocated on the $S_{m,n}$ which are already found above. If both do meet, the AHAP algorithm assumes allocating the v_i on the $S_{m,n}$ where the v_l is allocated, and calculate $EFT(v_i, S_{m,n})$ followed by marking the execution finish time as an *EFT2*. If the *EFT1* is smaller than the *EFT2*, which means moving the v_i does not reduce the execution finish time of the v_l, the v_i is allocated on the $S_{j,k}$ and the v_l is allocated on the $S_{m,n}$. But if the *EFT1* is larger than the *EFT2*, which means moving the v_i reduces the execution finish time of the v_l, both the v_i and the v_l are allocated on the $S_{m,n}$. Lastly, both the v_i and the v_l are removed from L. Algorithm 3 shows a pseudo code of the improved

insertion method.

Algorithm 3. The improved insertion method

A *improved_insertion_method(v_i, L)* procedure
  if next task v_l in the list L is *succ(v_i)* **do**
    Select an instance $S_{j,k}$ that minimizes *EFT(v_i, $S_{j,k}$)* with insertion
    Assume allocating v_i on $S_{j,k}$
    Calculate *EFT(v_l, $S_{m,n}$)* with insertion and mark as *EFT1*
    if v_i is *dpred(v_l)* and $S_{j,k} \neq S_{m,n}$ **do**
      Assume allocating v_i on $S_{m,n}$
      Calculate *EFT(v_l, $S_{m,n}$)* and mark it as an *EFT2*
      if *EFT1* < *EFT2* **do**
        Allocate v_i on $S_{j,k}$
        Allocate v_l on $S_{m,n}$
      else
        Allocate v_i on $S_{m,n}$
        Allocate v_l on $S_{m,n}$
      endif
    else
      Allocate v_i on $S_{j,k}$
      Allocate v_l on $S_{m,n}$
    endif
    Remove v_i and v_l from L
  else
    Select $S_{j,k}$ that minimizes *EFT(v_i, $S_{j,k}$)* with insertion
    Allocate v_i on $S_{j,k}$
    Remove v_i from L
  endif

5 Performance Evaluation

We have implemented a simulator to verify the performance of the proposed AHAP algorithm. The performance of the AHAP algorithm has been verified by applying the AHAP algorithm to existing scheduling heuristics. The graphs used in experiments are ones from STDGP (STanDard Task Graph Project) [17], and ones that model real world applications, such as FFT and GE. STDGP offers task graphs for fair performance comparison of the algorithms under the same conditions including task graphs generated from actual application programs.

The input parameters for simulations were set to the following values. There are 4 or 5 values in each parameter and by combining them, we were able to simulate various input task graphs and VM instances of cloud computing environment with 360,000 times of simulations.

The number of tasks in an input graph. *v*. The number of tasks in an input task graph. There are 180 graphs for each number of tasks which have various graph topologies. $v \in \{100, 300, 500, 750\}$

Communication to computation cost ratio. *ccr*. The ratio of the average communication cost of edges to the average computation cost of tasks in an input DAG. Low *ccr* indicates that an input DAG is computation intensive, and large *ccr* indicates that an input DAG is communication intensive. $ccr \in \{0.1, 0.5, 1.0, 2.5, 5.0\}$

Computation resource heterogeneity factor. β. The heterogeneity factor for speed of computational resources. Low heterogeneity indicates that a computation cost of a task is almost equal on each computational resource, and large heterogeneity indicates that a computation cost of a task is significantly different on each computational resource. $\beta \in \{0.1, 0.5, 1.0, 1.5, 2.0\}$

The number of provisioned VM instances. *vm*. The number of provisioned instances for execution of an application. $vm \in \{2, 4, 8, 16, 32\}$

Main comparison metric is an SLR (Schedule Length Ratio). The SLR is a normalized value of the total execution time of an input DAG, which is the total execution time divided by the sum of the minimum execution time of tasks on a critical path. The SLR can be expressed as an equation (10).

$$\text{SLR} = \frac{\textit{total execution time}}{\sum_{v_i \in CP} \min_{S_{j,k} \in S} \{w(v_i, S_{j,k})\}} \quad \textbf{(10)}$$

where *CP* refers to the critical path. The SLR of a graph cannot be less than one since the denominator is the lower bound. The algorithm which gives the lowest SLR is the best algorithm in terms of the performance.

The performance of the AHAP algorithm has been verified by applying the AHAP algorithm to the previous scheduling heuristics which aim to achieve the best performance in terms of execution time without regarding cost, such as the HCPFD, the HCT, and the PETS. The IC-PCP and the POSH are not included in the simulations because they aim to minimize user cost and as a result, there are significant performance gap between them. Since above scheduling heuristics are designed for a heterogeneous multiprocessor computing system, we have slightly modified and applied the algorithms to a cloud computing environment.

Figure 4 shows an average SLR with respect to the number of tasks in an input graph. For each *v*, we ran 22,500 times of simulations by combining *ccr*, β, and *vm*. The ones with the proposed AHAP algorithm outperforms the conventional algorithms in terms of an average SLR. An average SLR of the HCPFD algorithm with the AHAP algorithm is better than the conventional HCPFD algorithm by 6.31%, the HCT with the AHAP than the conventional HCT by 4.10%, and the PETS with the AHAP than the conventional PETS by 6.84%. As shown in Figure 4, the AHAP

algorithm provides better performance along with the increase of the number of tasks.

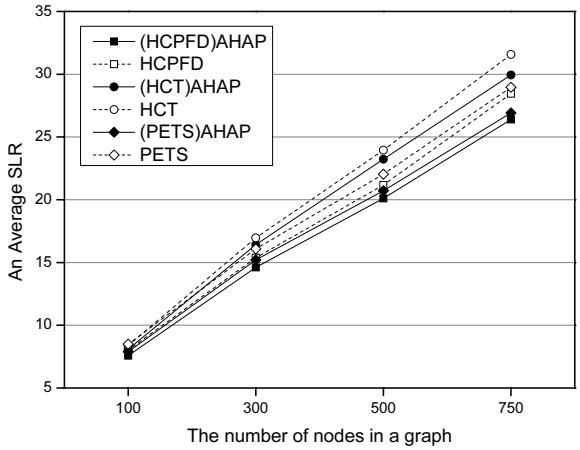

Figure 4. An average SLR with respect to *v*

Figure 5 shows an average SLR with respect to communication to computation cost ratio. For each *ccr*, we ran 18,000 times of simulations by combining *v*, *β*, and *vm*. The ones with the proposed AHAP algorithm outperforms the conventional algorithms in terms of an average SLR, on all *ccr* values. An average SLR of the HCPFD algorithm with the AHAP algorithm is better than the conventional HCPFD algorithm by 3.80%, the HCT with the AHAP than the conventional HCT by 2.79%, and the PETS with the AHAP than the conventional PETS by 5.51%. As shown in Figure 5, the AHAP algorithm shows better performance not only in computation intensive graphs but also in communication intensive graphs thanks to the selective resource allocation method and improved insertion method.

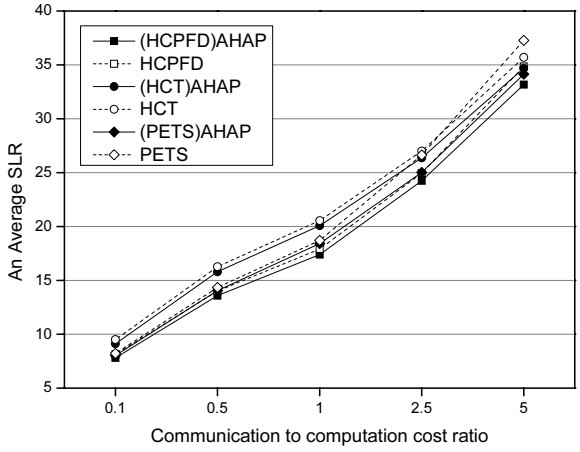

Figure 5. An average SLR with respect to *ccr*

Figure 6 shows an average SLR with respect to heterogeneity factor. For each *β*, we ran 18,000 times of simulations by combining *v*, *ccr* and *vm*. The ones with the proposed AHAP algorithm outperforms the conventional algorithms in terms of an average SLR regardless of heterogeneity factor. An average SLR of the HCPFD algorithm with the AHAP algorithm is better than the conventional HCPFD algorithm by 3.03%, the HCT with the AHAP than the conventional HCT by 2.83%, and the PETS with the AHAP than the conventional PETS by 6.12%. As shown in Figure 6, the AHAP algorithm shows better performance from low heterogeneity factor through high heterogeneity factor thanks to the selective resource allocation method and improved insertion method.

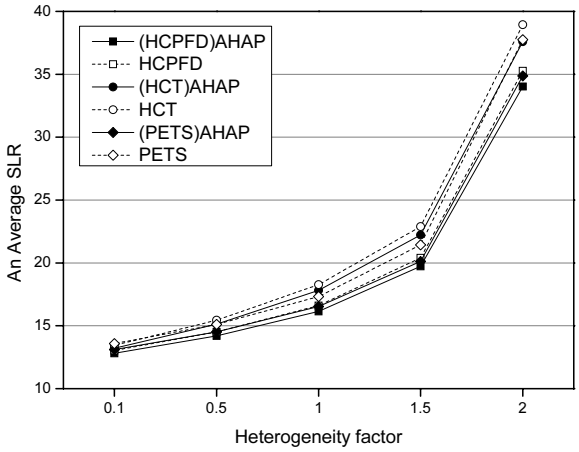

Figure 6. An average SLR with respect to *β*

Figure 7 shows an average SLR with respect to the number of provisioned VM instances. For each *vm*, we ran 18,000 times of simulations by combining *v*, *ccr*, and *β*. The ones with the proposed AHAP algorithm outperforms the conventional algorithms in terms of an average SLR. An average SLR of the HCPFD algorithm with the AHAP algorithm is better than the conventional HCPFD algorithm by 2.78%, the HCT with the AHAP than the conventional HCT by 1.46%, and the PETS with the AHAP than the conventional PETS by 5.26%. As shown in Figure 7, the efficiency of the algorithms decrease in environments with more than 8 instances. However, the AHAP algorithm provides better performance despite of decreased efficiency. And according to the figure, leasing 8 instances from the cloud provider when running applications with large scale data on the IaaS clouds might be the best choice considering both the performance and the cost.

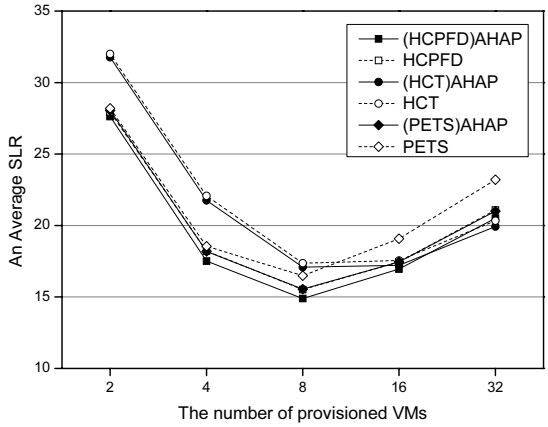

Figure 7. An average SLR with respect to *vm*

Figure 8 and Figure 9 shows an average SLR with respect to the number of data points in FFT graphs and with respect to matrix size in GE graphs, respectively. For each number of data points in FFT graphs and for each matrix size in GE graphs, we ran 125 simulations by combining *ccr*, β, and *vm*. As shown in Figure 8, the ones with the AHAP algorithm outperforms the conventional algorithms in terms of an average SLR on all data points and all matrix sizes. In Figure 8, the PETS algorithm with the AHAP algorithm shows 5.79% better performance when compared to the conventional one, and in Figure 9, the PETS algorithm with the AHAP algorithm shows 5.88% better performance than the conventional one. And especially, the PETS algorithm with the AHAP algorithm shows notable performance improvements in Figure 9. It can be analyzed that the notable performance improvements come from the duplications of tasks and proposed improved insertion method, of which the conventional algorithm does not utilize.

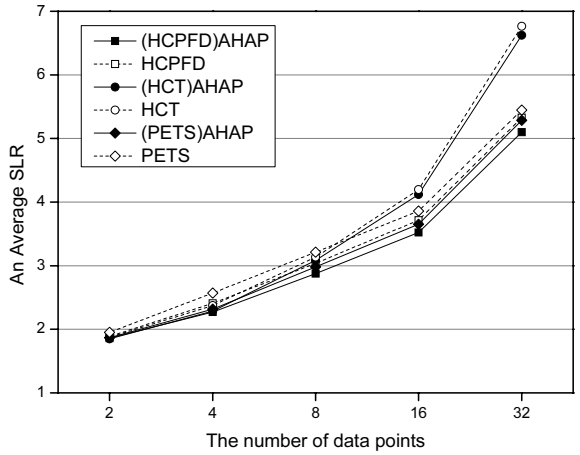

Figure 8. An average SLR with respect to the number of data points in FFT graaphs

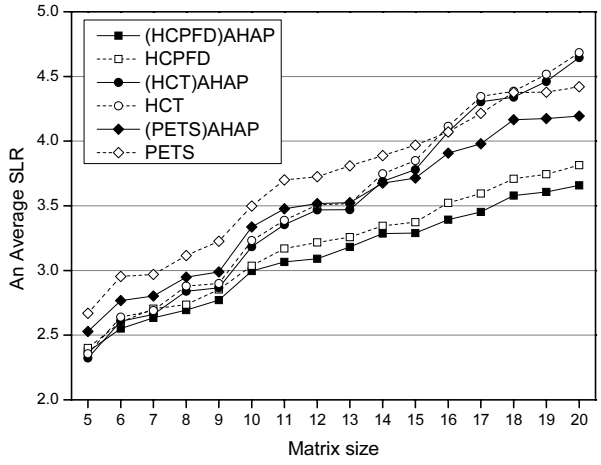

Figure 9. An average SLR with respect to matrix size in GE graphs

6 Conclusion

In this paper, we presented a novel resource allocation algorithm for cloud computing environment, called the AHAP, and an improved insertion method. Unlike conventional algorithms, the AHAP algorithm utilizes both a duplication method and an improved insertion method according to the characteristics of environment. The AHAP algorithm allocates a task onto a VM instance after evaluating the possibilities of reducing the execution time of the two methods based on the characteristics of an input DAG, a *ccr*, and a target cloud computing environment, a β. And the improved insertion method allocates a task onto a VM instance with an insertion method while considering an execution finish time of a child task. The simulations with various parameters including multiple *ccr* and β shows that the proposed AHAP algorithm provides better performance than conventional algorithms. Further researches will be on finding more efficient criterion for choosing an allocation method with dynamic scaling of the number of VM instances.

Acknowledgement

This work was supported by INHA UNIVERSITY Research Grant.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A View of Cloud Computing, *Communications of the ACM*, Vol. 53, No. 4, pp. 50-58, April, 2010.

[2] Q. Zhang, L. Cheng, R. Boutaba, Cloud Computing: State-of-the-art and research challenges, *Journal of Internet Services and Applications*, Vol. 1, No. 1, pp. 7-18, May, 2010.

[3] G. Pallis, Cloud Computing: The New Frontier of Internet Computing, *IEEE Internet Computing*, Vol. 14, No. 5, pp. 70-73, September-October, 2010.

[4] R. Van den Bossche, K. Vanmechelen, J. Broeckhove, Online Cost-efficient Scheduling of Deadline-constrained Workloads on Hybrid Clouds, *Future Generation Computer Systems*, Vol. 29, No. 4, pp. 973-985, June, 2013.

[5] L.-Y. Tseng, S.-S. Wang, S.-C. Wang, K.-Q. Yan, Essentiality of Deadline for Task Scheduling in Cloud Computing, *Journal of Internet Technology*, Vol. 16 No. 1, pp. 47-60, January, 2015.

[6] J. H. Park, J. C. Hung, N. Y. Yen, Y.-S. Jeong, Guest Editorial: Advanced Convergence Technologies: Big Data, IoT, Cloud Computing, *Journal of Internet Technology*, Vol. 15 No. 4, pp. 589-591, July, 2014.

[7] S. Akioka, Y. Muraoka, HPC Benchmarks on Amazon EC2, *IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, Perth, Australia, 2010, pp. 1029-1034.

[8] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, P. Maechling, Scientific Workflow Applications on Amazon EC2, *IEEE 5th International Conference on E-Science Workshops*, Oxford, UK, 2009, pp. 59-66.

[9] O. Sinnen, *Task Scheduling for Parallel Systems*, Wiley, 2007.

[10] T. Hagras, J. Janecek, A High Performance, Low Complexity Algorithm for Compile-time Task Scheduling in Heterogeneous Systems, *Parallel Computing*, Vol. 31, No. 7, pp. 653-670, July, 2005.

[11] L. Zhou, S. Shixin, Scheduling Algorithm Based on Critical Tasks in Heterogeneous Environments, *Journal of Systems Engineering and Electronics*, Vol. 19, No. 2, pp. 398-405, April, 2008.

[12] E. Ilavarasan, P. Thambidurai, R. Mahilmannan, Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System, *4th International Symposium on Parallel and Distributed Computing*, Lille, France, 2005, pp. 28-38.

[13] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, pp. 260-274, March, 2002.

[14] S. Abrishami, M. Naghibzadeh, D. H. J. Epema, Deadline-constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds, *Future Generation Computer Systems*, Vol. 29, No. 1, pp. 158-169, January, 2013.

[15] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, J. Wang, Cost-efficient Task Scheduling for Executing Large Programs in the Cloud, *Parallel Computing*, Vol. 39, No. 4-5, pp. 177-188, April- May, 2013.

[16] R. N. Calheiros, R. Buyya, Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 7, pp. 1787-1796, July, 2014.

[17] http://www.kasahara.elec.waseda.ac.jp.

Biographies

Inseong Song received M.S. degree in electronic engineering at the Inha University, Korea, in 2011. He is currently in Ph.D. course at the Inha University. His research interests are parallel and distributed computing, computer architecture, and computer networks.

Jinhyuk Kim received M.S. degree in electronic engineering at the Inha University, Korea, in 2011. He is currently in Ph.D. course at the Inha University. His research interests are computer networks, computer architecture, and parallel and distributed computing.

Sangbang Choi received Ph.D. in computer science at the University of Washington, U.S., in 1988. He is a professor in department of electronic engineering at the Inha University. His research interests are computer architecture, computer networks, and parallel and distributed computing.