Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 237960, 10 pages
http://dx.doi.org/10.1155/2014/237960

Research Article

Task Balanced Workflow Scheduling Technique considering Task Processing Rate in Spot Market

Daeyong Jung,[1] JongBeom Lim,[1] JoonMin Gil,[2] Eunyoung Lee,[3] and Heonchang Yu[1]

[1] *Department of Computer Science Education, Korea University, 321A, Lyceum, Anam-Dong, Seongbuk-Gu, Seoul 136-701, Republic of Korea*
[2] *School of Information Technology Engineering, Catholic University of Daegu, Daegu, Republic of Korea*
[3] *Department of Computer Science, Dongduk Women's University, Seoul, Republic of Korea*

Correspondence should be addressed to Heonchang Yu; yuhc@korea.ac.kr

Received 21 January 2014; Accepted 4 June 2014; Published 29 June 2014

Academic Editor: Young-Sik Jeong

Recently, the cloud computing is a computing paradigm that constitutes an advanced computing environment that evolved from the distributed computing. And the cloud computing provides acquired computing resources in a pay-as-you-go manner. For example, Amazon EC2 offers the Infrastructure-as-a-Service (IaaS) instances in three different ways with different price, reliability, and various performances of instances. Our study is based on the environment using spot instances. Spot instances can significantly decrease costs compared to reserved and on-demand instances. However, spot instances give a more unreliable environment than other instances. In this paper, we propose the workflow scheduling scheme that reduces the out-of-bid situation. Consequently, the total task completion time is decreased. The simulation results reveal that, compared to various instance types, our scheme achieves performance improvements in terms of an average combined metric of 12.76% over workflow scheme without considering the processing rate. However, the cost in our scheme is higher than an instance with low performance and is lower than an instance with high performance.

1. Introduction

In recent years, due to the increased interests in cloud computing, many cloud projects and commercial systems such as Amazon EC2 [1] have been implemented. Cloud computing provides many benefits including easy access to user data, ease of management for users, and the reduction of costs. And cloud computing services provide a high level of scalability of computing resources combined with internet technology to many customers [2, 3]. In most cloud services, the concept of an instance unit is used to provide users with resources in a cost-efficient manner. There are many different cloud computing providers and each offers different layers of services. This paper focuses on Infrastructure-as-a-Service (IaaS) platforms that allow clients access to massive computational resources in the form of instances [4–7].

Generally, cloud computing resources use reliable on-demand instances. On-demand instances allow the user to pay for computing capacity by hour, with no long-term commitments. This frees users from the costs and complexities of planning, purchasing, and maintaining hardware and transforms what are usually large fixed costs into much smaller variable costs [1]. However, on-demand instance may incur upper cost than other instances such as reserved instance and spot instance. We focus on spot instances in unreliable environment. For such a reason, if you have time flexibility for executing applications, spot instances can significantly decrease your Amazon EC2 costs [8, 9]. For task completion, therefore, spot instances may incur lower cost than on-demand instances.

The spot instance is configured by spot market-based cloud environment. In the spot instance environment, variations of spot prices are dependent on the supply and demand of spot instances. The environment affects the successful completion or failure of tasks depending on the variation of spot prices. Spot prices have a market structure and follow the law of demand and supply. Therefore, cloud services (Amazon EC2) can provide a spot instance when a user's bid is

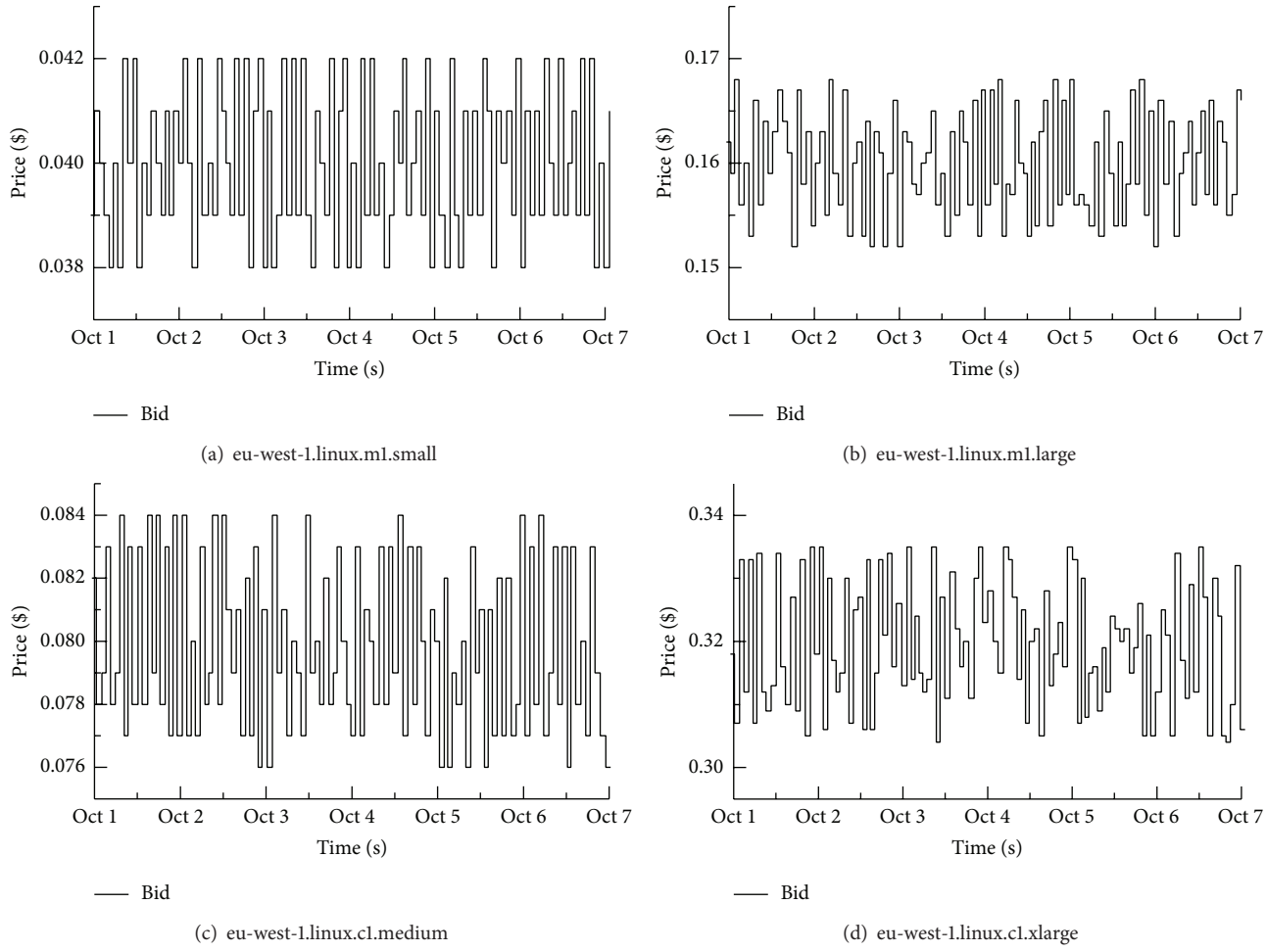


Figure 1: Price history of EC2's spot instances.

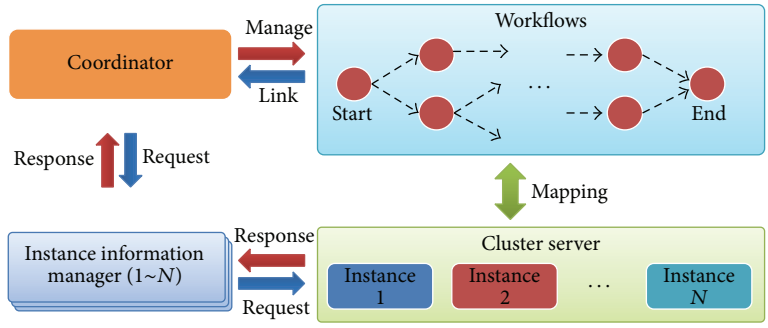
higher than the current spot price. Further, a running instance stops when a user's bid becomes less than or equal to the current spot price. After a running instance stops, it restarts when a user's bid becomes greater than the current spot price [10–12].

In particular, the scientific application makes the current common of workflow. However, the spot instance-based cloud computing takes various performances. In spot instance, an available execution time depends on a spot price. The spot price changes periodically based on user's demand and supply. The completion time for the same amount of a task varies according to the performance of an instance. In particular, the failure time of each instance differs according to the user's bid and the performance in an instance. Therefore, we solve the problem that a completion time of a task in an instance increases when a failure occurs. For an efficient execution of a task, the task is divided into subtasks on various types of available instances. We analyze information of the task and the instance from price history. We estimate the size of task and the information of an available instance from the analyzed data. We create workflow using each available instance and the size of a task. As a consequence, we propose the scheduling scheme using workflow to solve job execution problem and considering task processing rate. And we execute user's job at the boundary of selected instances and expand the suggested user budget.

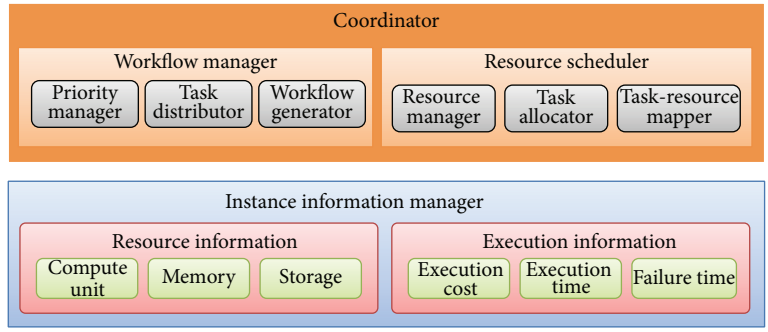
2. Background and Related Works

In this section, we begin by describing the workflow model focusing on spot instances. Firstly, we explain the background of spot instances in cloud environments. In the spot instances environment [8, 9], there are numerous studies on fault tolerance [10–12] and workflow scheduling [13, 14].

2.1. Spot Instances. Amazon EC2 offers the IaaS instances in three different ways with different price, reliability, and various performances of instances. Those are reserved instances, on-demand instances, and spot instances. In case of reserved instances, a user pays a yearly fee and receives a discount on hourly rates. And, in case of on-demand instances, a user pays the fee on hourly rate. In spot instances, a user determines the user's bid and spot price decides spot market based on the user's demand and supply. Our scheduling focuses on offering



(a) The mapping relation of workflow and instances



(b) The constitution of coordinator and manager

Figure 2: Workflow environment.

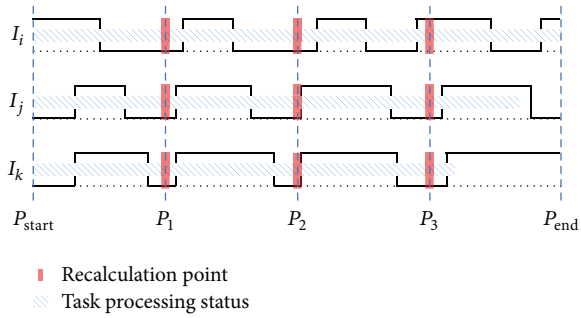


Figure 3: The recalculation point of the task size.

services at the boundary of spot instances. Spot instances give an unreliable environment compared to reserved and on-demand instances. However, spot instances can significantly decrease user's costs compared to other instances. The spot price in spot instance is based on market structure and law of demand and supply. Therefore, cloud service can provide a spot instance when a user's bid is higher than the current spot price. If the user's bid exceeds the current market price, the user runs the instance. However, if the market price exceeds the user's bid, the instance is terminated and the partial hours are not charged. And the spot system immediately stops the spot instance without any notice to the user. After a running instance stops, the instance restarts when a user's bid is greater than the current spot price. An example of spot history is shown in Figure 1. This figure shows examples of fluctuations of spot price for standard instance (m1-small and m1-large) and high-CPU instance (c1-medium and c1-xlarge) during 7 days in October 2010 [15].

2.2. Fault Tolerance. On the fault tolerance side, two similar studies (hour-boundary checkpointing [10] and rising edge-driven checkpointing [11]) proposed enforcing fault tolerance in cloud computing with spot instance. Based on the actual price history of EC2 spot instances, they compared several adaptive checkpointing schemes in terms of monetary costs and job execution time. In hour-boundary checkpointing, the checkpointing operation is performed in the hour boundary, and a user pays the biding price on an hourly basis. In rising edge-driven checkpointing, checkpointing operation is performed when the price of the spot instance is raised and the price is less than the user's bid. However, two schemes have problems that the costs and task completion time are increased due to increase of the number of checkpoints. To solve these problems, in our previous study [12], we proposed the checkpointing scheme using checkpoint thresholds based on rising-driven checkpointing. The checkpointing is basically performed using two thresholds, price and time, based on the expected execution time according to the price history. Therefore, we propose a workflow system to apply the previous proposed checkpointing.

2.3. Workflow Scheduling. A workflow is a model that represents complex problems with structures such as directed acyclic graphs (DAG). Workflow scheduling is a kind of

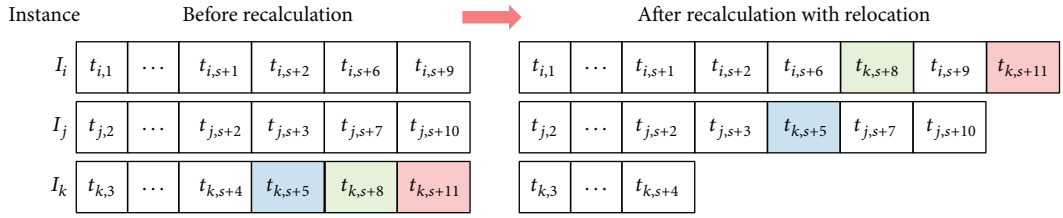


Figure 4: The recalculation operation of the assigned task.

global task scheduling as it focuses on mapping and managing the execution of interdependent tasks on shared resources. However, the existing workflow scheduling methods have the limited scalability and are based on centralized scheduling algorithm. Consequently, these methods are not suitable for spot instance-based cloud computing. In spot instance, the job execution has to consider available time and cost of an instance. Fully decentralized workflow scheduling system determines the instance to use the chemistry-inspired model in community cloud platform [13]. A throughput maximization strategy is designed for transaction-intensive workflow scheduling that does not support multiple workflows [14]. Our proposed scheduling guarantees an equal task distribution to available instances in spot instance-based cloud computing. And the scheduling method performs redistribution of the tasks based on task processing rate.

3. Proposed Workflow System

3.1. System Architecture. Our proposed scheme is expanded from our previous work [12] and includes a workflow scheduling algorithm. Figure 2(a) presents the relation of workflows and instances and Figure 2(b) shows the constitution of coordinator and manager. Figure 2 illustrates the roles of the instance information manager, the workflow manager, and the resource scheduler. The instance information manager obtains information for the job allocation and resource management. The information includes VM specifications in each instance and the execution-related information such as the execution costs, execution completion time, and failure time. The execution-related information is calculated by using the selected VM based on spot history. The workflow manager and resource scheduler extract the needed execution-related information from the instance information manager. Frist, the workflow manager generates the workflow for the requested job. The generated workflow determines the task size according to the VM performance, the execution time and costs, and the failure time when the selected instance is used. Secondly, the resource scheduler manages the resource and allocates the task to handle the job. Resource and task managements are needed in order to reallocate tasks when the resource cannot get the information for the task and when the task has a fault during execution.

3.2. Workflow Scheduling Technique considering Task Processing Rate. The scheduling scheme is depicted in Figure 3. The instances I_i, I_t, and I_k mean high, medium, and low performance, respectively. The instance I_k belongs to a positive group and the other two instances (I_i, I_j) belong to a negative group. The scheduler distributes a task size to allocate available instances and considers performance of instances. Task size recalculation points divide the fourth quarter based on the expected task execution time and recalculate each quarter except for the last quarter. The task size rate is determined based on the average of task execution time of each instance within the recalculated point. And the modified task size in each instance is allocated to consider the task size rate.

Figure 4 shows the recalculation point of the task size from the P_1 position in Figure 3. In Figure 4, we assume that the processing rate of instances is proportional to the performance of instances. The left side of Figure 4, "before recalculation," represents the tasks assigned to each instance. The right side, "after recalculation with relocation," shows the result of task migration based on the average task execution time in each instance. After a recalculation operation, we perform the rearrangement of tasks. The rearrangement method sorts tasks in increasing order of their indices.

To design the above model, our proposed scheme uses the workflow in spot instance and its purpose is to minimize job processing time within the suggested cost of user. The task size is determined by considering the availability and performance of each instance in order to minimize the job processing time. The available time is estimated by the execution time and cost using the price history of spot instances to improve the performance and stability of task processing. The estimated data is determined to assign the amount of tasks to each instance. Our proposed scheme reduces the out-of-bid situation and improves the job execution time. However, total cost is higher than when not using workflow.

Our task distribution method determines the task size in order to allocate a task to a selected instance. Based on a compute-unit and an available state, the task size of an instance $I_i(T_{I_i})$ is calculated as follows:

$$T_{I_i} = \left(\frac{U_{I_i} \times A_{I_i}}{\sum_{i=1}^{N} \left(U_{I_i} \times A_{I_i} \right)} \right) \times \frac{1}{U_{I_i}} \times T_{\text{request}} \times U_{\text{baseline}}, \quad (1)$$

where T_{request} represents the total size of tasks required for executing a user request. In an instance I_i, U_{I_i} and A_{I_i} represent the compute-unit and the available state, respectively. The available state A_{I_i} can be either 0 (unavailable) or 1 (available).

```
(1) Boolean S_flag = false // a flag representing occurrence of a task execution
(2) while (search user's job) do
(3)   if (require job execution by the user) then
(4)     take the cost and total execution time by the user;
(5)     S_flag = true;
(6)   end if
(7)   if (S_flag) then
(8)     invoke initial_workflow ( );  // thread function
(9)   while (task execution does not finish) do
(10)      if (meet the recalculation point by instance)  then
(11)        invoke recalculation_workflow ( );  // thread function
(12)      end if
(13)  end while
(14)  end if
(15) end while
```

Algorithm 1: Workflow scheduling algorithm.

```
(1) Thread_Function initial_workflow ( ) begin
(2)   forall instance I_i ∈ Ins do
(3)     retrieve an instance information to meet the user's requirement in an
        instance I_i;
(4)     analyze an available execution time and cost in an instance I_i;
(5)     store the analyzed available instance to a queue_instance;
(6)   end forall
(7)   calculate on priority list for the priority job allocation;
(8)   forall instance I_i ∈ queue_instance do
(9)     allocate tasks to the instance I_i;
(10)  end forall
(11) end Thread_Function
(12) Thread_Function recalculation _workflow ( ) begin
(13)  forall instance I_i ∈ Ins do
(14)    retrieve the information T_rate^{I_i} to an instance I_i;
(15)    calculate the modified task size;
(16)  end forall
(17) end Thread_Function
```

Algorithm 2: Workflow recalculation function.

We use the instance rate $T_{\text{rate}}^{I_i}$ for determining the criteria to divide groups. $T_{\text{rate}}^{I_i}$ represents the unit taken for the processing of a task size in the instance I_i. Consider

$$T_{\text{rate}}^{I_i} = \frac{T_{\text{execution}}^{I_i} + T_{\text{failure}}^{I_i}}{T_{\text{execution}}^{I_i}}, \tag{2}$$

where $T_{\text{execution}}^{I_i}$ and $T_{\text{failure}}^{I_i}$ represent the task execution time and the task failure time, respectively.

And we define the avg to classify groups. The avg is the average of available instances such as $T_{\text{rate}}^{\text{avg}}$ and T_{avg} which represent the average of the $T_{\text{rate}}^{I_i}$ and T_{I_i}, respectively. The set of instances is classified into two groups, positive and negative, based on $T_{\text{rate}}^{\text{avg}}$. The positive group G_P is the set of instances with $T_{\text{rate}}^{I_i}$ greater than $T_{\text{rate}}^{\text{avg}}$. Consider

$$G_P = \left\{ I_i T_{\text{rate}}^{I_i} \geq T_{\text{rate}}^{\text{avg}}, 1 \leq i \leq N \right\}. \tag{3}$$

We calculate the task size to transfer from instance I_i (Tr_{I_i}) in G_P as follows:

$$Tr_{I_i} = \frac{\left[\left(T_{I_I} - T_{\text{execution}}^{I_i} \right) \times I_{\text{rate}}^{I_i} - \left(T_{\text{avg}} - T_{\text{execution}}^{\text{avg}} \right) \times I_{\text{rate}}^{\text{avg}} \right]}{I_{\text{rate}}^{I_i}}, \quad 1 \leq i \leq N. \tag{4}$$

In group G_P, the task size of each instance I_i is given as $T'_{I_i \in G_P}$. We are able to get $T'_{I_i \in G_P}$ by considering Tr_{I_i} after the transfer operation:

$$T'_{I_i \in G_P} = T_{I_i \in G_P} - Tr_{I_i}, \quad 1 \leq i \leq N. \tag{5}$$

The negative group G_N is the set of instances I_i with $T_{\text{rate}}^{I_i}$ less than $T_{\text{rate}}^{\text{avg}}$. Consider

$$G_N = \left\{ I_i T_{rate}^{I_i} < T_{\text{rate}}^{\text{base}}, 1 \leq i \leq N \right\}. \tag{6}$$

Table 1: Information of resource types.

Instance type name	Compute unit	Virtual cores	Spot price min	Spot price average	Spot price max
m1.small (Standard)	1 EC2	1 core (1 EC2)	\$0.038	\$0.040	\$0.053
m1.large (Standard)	4 EC2	2 cores (2 EC2)	\$0.152	\$0.160	\$0.168
m1,xlarge (Standard)	8 EC2	4 cores (2 EC2)	\$0.076	\$0.080	\$0.084
c1.medium (High-CPU)	5 EC2	2 cores (2.5 EC2)	\$0.304	\$0.323	\$1.52
c1.xlarge (High-CPU)	20 EC2	8 cores (2.5 EC2)	\$0.532	\$0.561	\$0.588
m2.xlarge (High-Memory)	6.5 EC2	2 cores (3.25 EC2)	\$0.532	\$0.561	\$0.588
m2.2xlarge (High-Memory)	13 EC2	4 cores (3.25 EC2)	\$0.532	\$0.561	\$0.588
m2.4xlarge (High-Memory)	26 EC2	8 cores (3.25 EC2)	\$1.064	\$1.22	\$1.176

Table 2: Parameters and values for simulation.

Simulation parameter	Task time interval	Baseline	Distribution time	Merge time	Checkpoint time	Recovery time
Value	43,200 (s)	m1.xlarge	300 (s)	300 (s)	300 (s)	300 (s)

The tasks are allocated according to the instance performance U_{I_i}. The task size to receive R_{I_i} is allocated according to the task size of each instance I_i. In the group G_N, the task size of each instance is given as $T'_{I_i \in G_N}$. After the receive operation, R_{I_i} is added to $T_{I_i \in G_N}$. Consider

$$R_{I_i} = \frac{U_{I_i}}{\sum_{i \in G_N} U_{I_i}} \times \sum_{i \in G_p} \left(Tr_{I_i} \times U_{I_i}\right) \times \frac{1}{U_{I_i}}, \quad 1 \le i \le N, \tag{7}$$

$$T'_{I_i \in G_N} = T_{I_i \in G_N} + R_{I_i}, \quad 1 \le i \le N.$$

We propose a workflow scheduling algorithm based on the above equations. Algorithms 1 and 2 show the workflow scheduling algorithm and the workflow recalculation function, respectively.

4. Performance Evaluation

The simulations were conducted using the history data obtained from Amazon EC2 spot instances [15]. The history data before 10-01-2010 was used to extract the expected execution time and failure occurrence probability for our checkpointing scheme. The applicability of our scheme was tested using the history data after 10-01-2010.

In the simulations, one type of spot instance was applied to show the effect of an analysis—task time—on the performance. Table 1 shows various resource types used in Amazon EC2. In this table, resource types comprise a number of different instance types. First, standard instances offer a basic resource type. Second, high-CPU instances offer more compute-units than other resources and can be used for compute-intensive applications. Finally, high-memory instances offer more memory capacity than other resources and can be used for high-throughput applications, including database and memory caching applications. Under the simulation environments, we compare the performance of our proposed scheme with that of the existing schemes without distributions of tasks in terms of various analyses according to the task time.

Table 1 shows various information of resource type in each instance and Table 2 shows the parameters and values for simulation. The information of spot price is extracted from 11-30-2009 to 01-23-2011 in spot history. The user's bid is taken by the spot price average from information of spot price. The task size is decided by compute-unit rate based on baseline. Initially, the baseline denotes an instance m1.xlarge. For example, the task size of an instance m1.small is calculated by the following:

$$T_{\text{m1.small}} = \frac{U_{\text{m1.xlarge}}}{U_{\text{m1.small}}} \times T_{\text{original task}}. \tag{8}$$

4.1. Comparison Results of Each Instance before Applying Workflow. Figure 5 shows the simulation results about each instance. We consider performance condition of each instance. Each instance sets user's bid to take the spot price average in Table 2. Figure 5 presents the execution time and costs according to various instances types. The instance with high performance reduces the execution time but spends higher cost than the instance with low performance.

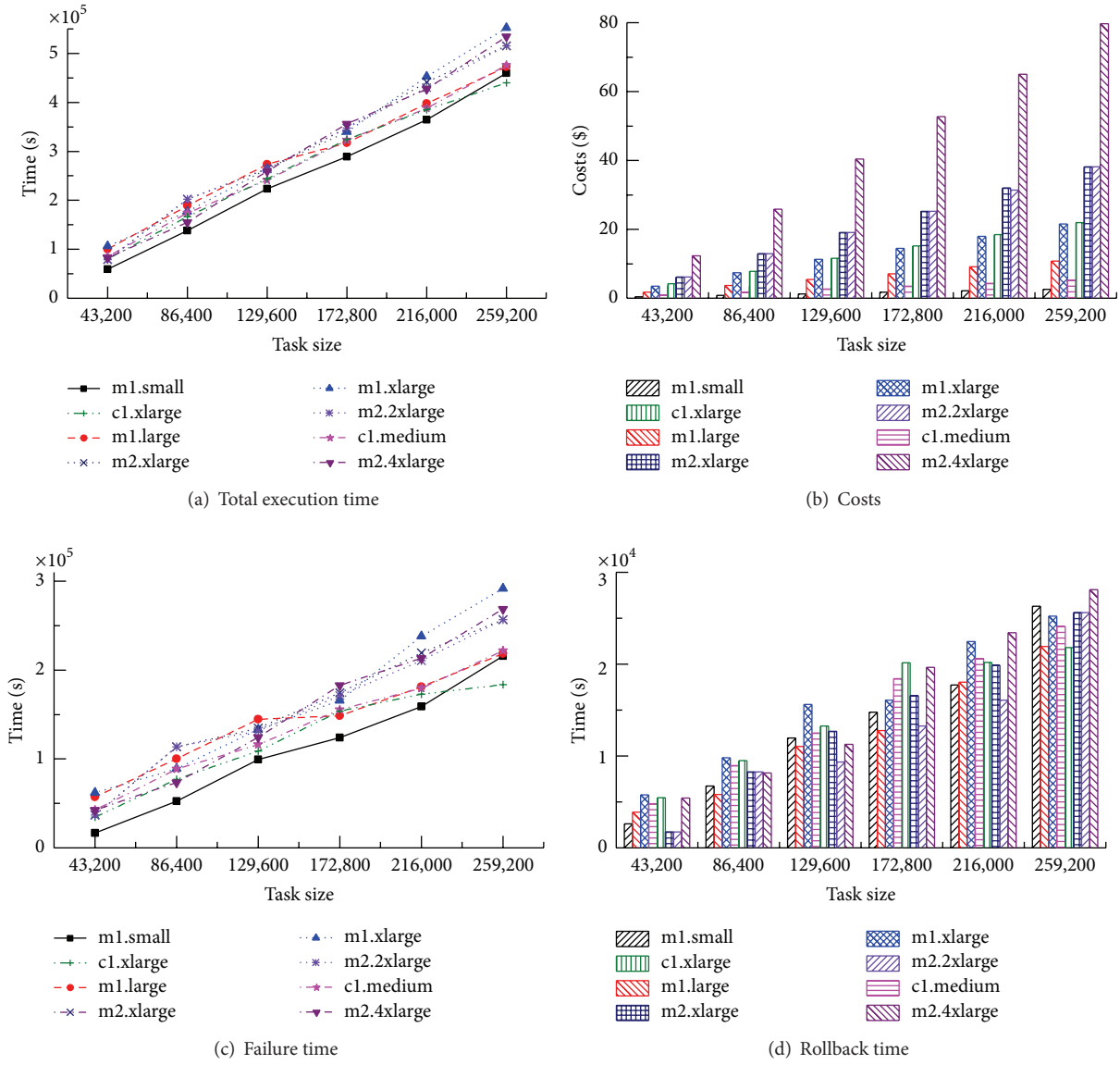


Figure 5: Simulation result in each instance.

As, in Figure 5(a), the total execution time increases, Figure 5(c) describes that the failure time increases. Figure 5(d) shows the rollback time in each instance. Rollback time is the time interval between a failure occurrence time and the last checkpoint time.

4.2. Comparison Results after Applying Workflow. Figure 6 shows the simulation results about the task distribution. Figure 6(a) shows the total execution time for each instance and Total. In the figures, Total_T denotes the total time taken for distributing and merging tasks. Total_C denotes the sum of costs of task execution in each instance. The total execution time of the Total_T achieves performance improvements in terms of an average execution time of 81.47% over the shortest execution time in each task time interval. In Figure 6(b), the cost in our scheme increases an average of \$11.64 compared to an instance m1.small and reduces an average of \$32.87 compared to an instance m2.4xlarge. A failure time of Figure 6(c) and a rollback time of Figure 6(d) are smaller than those of Figures 5(c) and 5(d).

Figure 7 shows the execution results of workflow based on the task processing rate after applying our proposed scheme. Figures 6(a) and 7(a) show that the total execution time is reduced by an average of 18.8% after applying our scheme compared to not applying it. Figures 6(c) and 7(c) show that the failure time after applying our proposed scheme was increased by 6.68% compared to before applying it. However, in Figures 6(d) and 7(d), the rollback time after applying our proposed scheme showed an average performance improvement of 4.3% when compared to the rollback time without applying it. The rollback time is calculated from a failure point to the last checkpoint time. Figures 6(b) and 7(b) show that

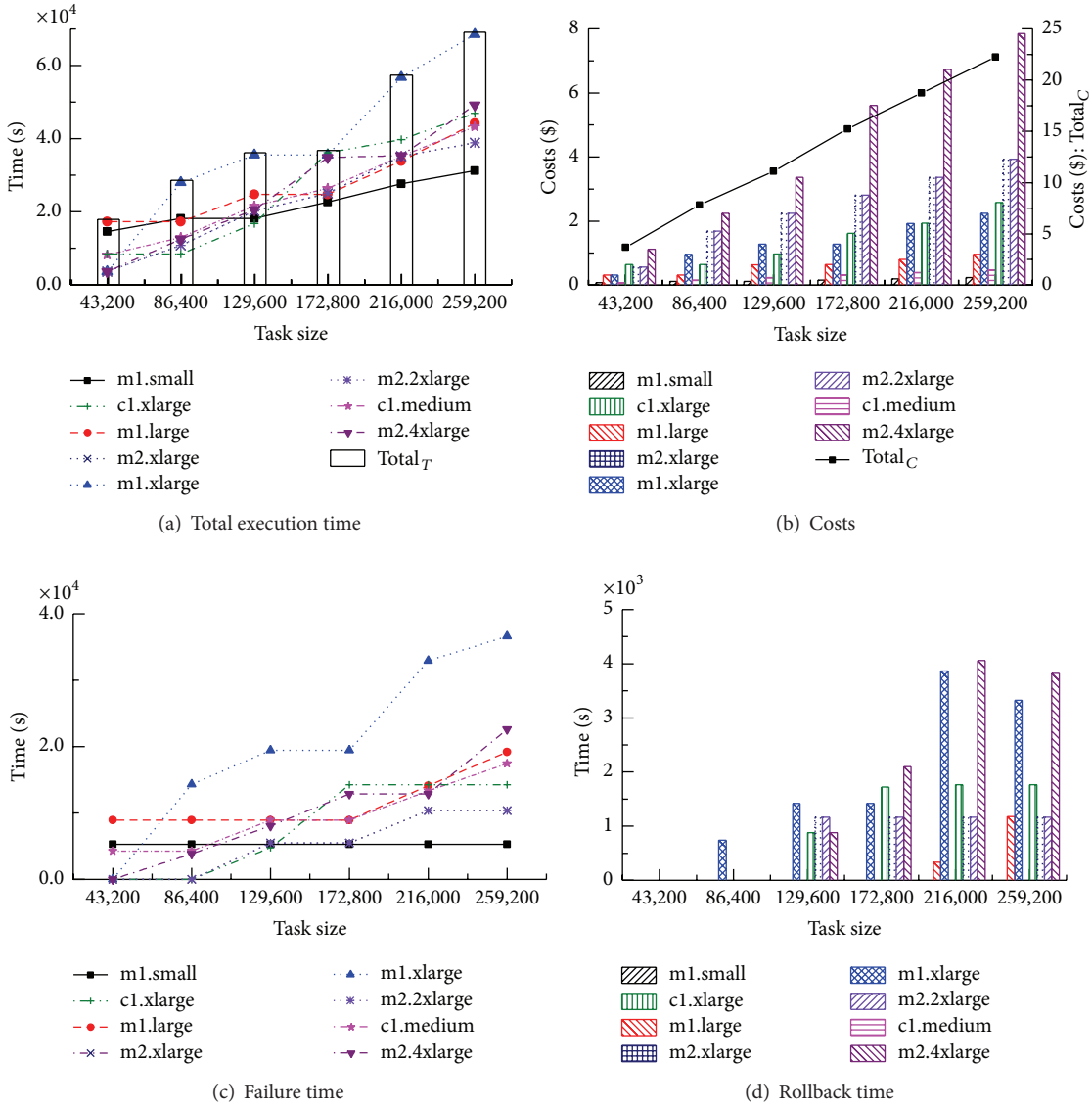


Figure 6: Simulation result in task distribution.

the total costs after applying our scheme decreased by an average of $0.37 when compared to the cost before applying it. There are two facts deduced from these results. One is the increase of failure time. The other is the improvement of total execution time through an efficient task distribution. And the task execution loss was reduced when the out-of-bid situation occurred. In addition, we compare experiments to consider the execution time and costs.

Figure 8 shows the combined performance metric and the product of the total task execution time and cost. According to the task time interval, there is a little difference between the basic and the applying schemes, compared to each instance. In the figure, the basic scheme denotes the workflow product that applies only task distribution without considering a task processing rate. The applying scheme denotes the workflow product considering the task processing rate. The product of the basic scheme achieves performance improvements in the average combined metric of 87.71% over the average product instance in each task time interval. The applying scheme achieves performance improvements in the average combined metric of 12.76%, compared to the basic scheme.

5. Conclusion

In this paper, we proposed a workflow scheduling technique considering task processing rate in unreliable cloud computing environments. The workflow scheduling scheme recalculates the task size based on task processing rate within the recalculated point. In addition, our previously proposed checkpoint scheme takes a checkpointing based on two kinds of thresholds: price and time. Our scheme reduces a failure time and an absolute time through the checkpoint scheme. The rollback time of our scheme can be less than that of

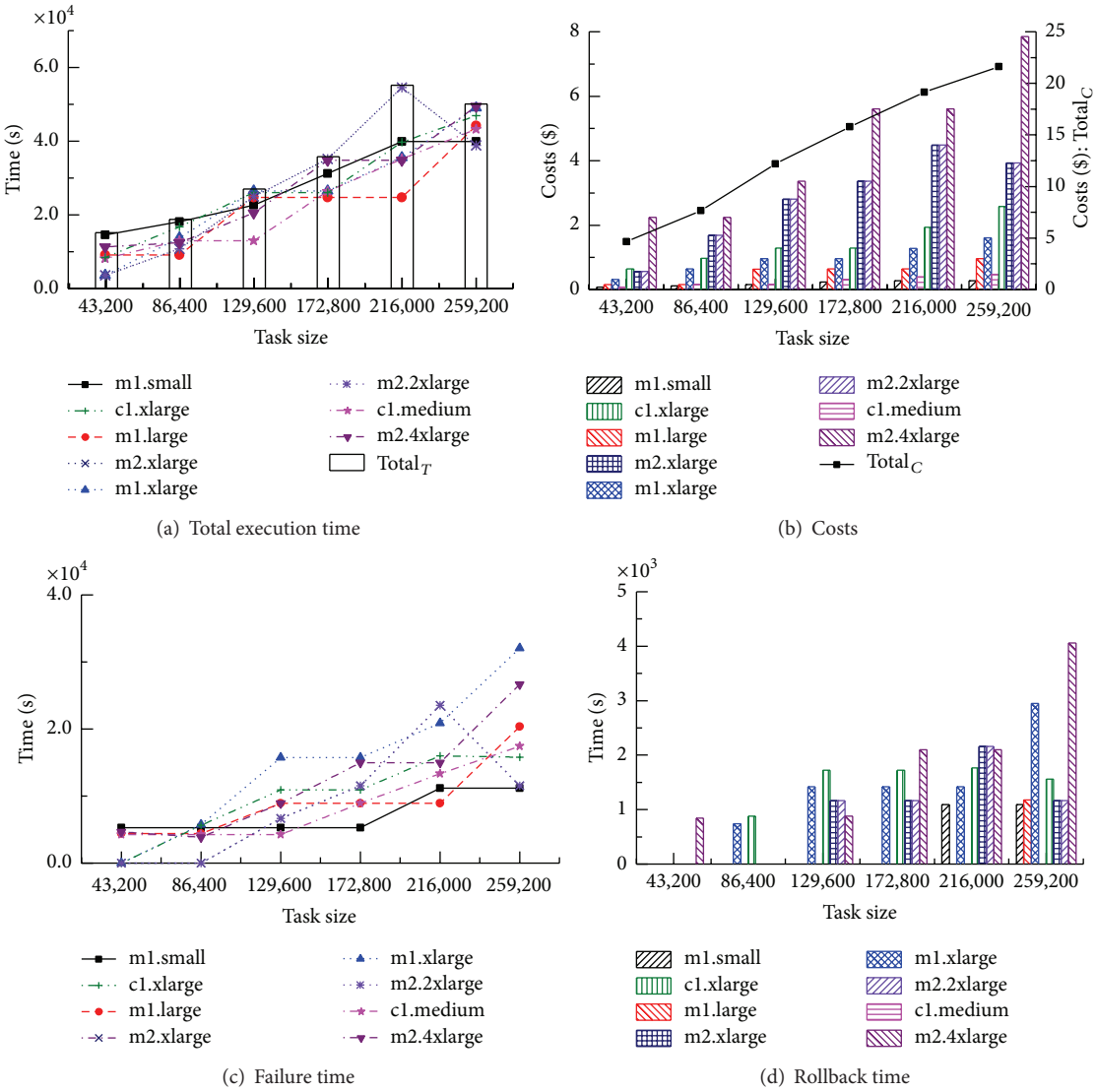


Figure 7: Simulation result in task distribution considering task processing rate.

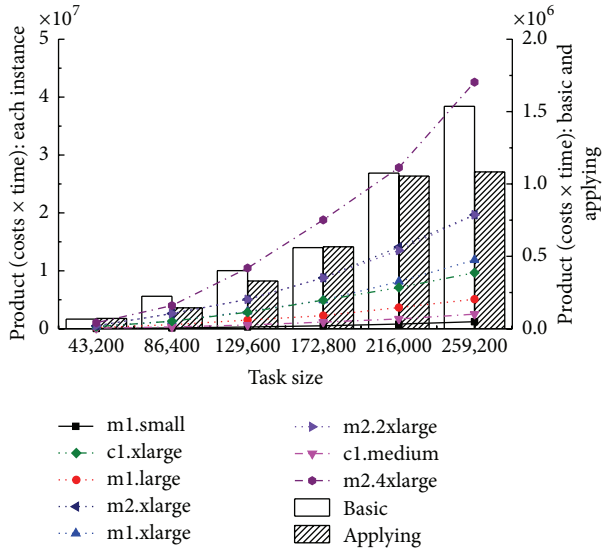


Figure 8: Comparison of combined metrics (total task execution time and cost).

the existing scheme without workflow because our scheme adaptively performs task distribution operation according to available instances. The simulation results showed that the average execution time in our scheme was improved by 17.8% after applying our proposed scheme as compared to before applying it. And our proposed scheme represented approximately the same cost as compared to before applying it. Other simulation results reveal that, compared to various instance types, our scheme achieves performance improvements in terms of an average combined metric of 12.76% over workflow scheme without considering task processing rate.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (NRF-2012R1A2A2A02046684).

References

[1] *Elastic Compute Cloud (EC2)*, 2013, http://aws.amazon.com/ec2.

[2] H. N. Van, F. D. Tran, and J.-M. Menaud, "SLA-aware virtual resource management for cloud infrastructures," in *Proceedings of the 9th IEEE International Conference on Computer and Information Technology (CIT '09)*, vol. 1, pp. 357–362, Xiamen, China, October 2009.

[3] K. Mahajan, A. Makroo, and D. Dahiya, "Round robin with server affinity: a VM load balancing algorithm for cloud based infrastructure," *Journal of Information Processing System*, vol. 9, no. 3, pp. 379–394, 2013.

[4] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC '08)*, pp. 5–13, Dalian, China, September 2008.

[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE '08)*, pp. 1–10, Austin, Tex, USA, November 2008.

[6] M. M. Weng, T. K. Shih, and J. C. Hung, "A personal tutoring mechanism based on the cloud environment," *Journal of Convergence*, vol. 4, no. 3, pp. 37–44, 2013.

[7] A. Følstad, K. Hornbæk, and P. Ulleberg, "Social design feedback: evaluations with users in online ad-hoc groups," *Human-Centric Computing and Information Sciences*, vol. 3, article 18, 2013.

[8] *Amazon EC2 Spot Instances*, 2013, http://aws.amazon.com/ec2/spot-instances/.

[9] SpotCloud, 2014, http://www.spotcloud.com.

[10] S. Yi, J. Heo, Y. Cho, and J. Hong, "Taking point decision mechanism for page-level incremental checkpointing based on cost analysis of process execution time," *Journal of Information Science and Engineering*, vol. 23, no. 5, pp. 1325–1337, 2007.

[11] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD '10)*, pp. 236–243, July 2010.

[12] D. Jung, S. Chin, K. Chung, H. Yu, and J. Gil, "An efficient checkpointing scheme using price history of spot instances in cloud computing environment," in *Proceedings of the 8th IFIP International Conference on Network and Parallel Computing*, vol. 6985 of *Lecture Notes in Computer Science*, pp. 185–200, 2011.

[13] H. Fernandez, M. Obrovac, and C. Tedeschi, "Decentralised multiple workflow scheduling via a chemically-coordinated shared space," Research Report RR-7925, Inria, 2012.

[14] K. Liu, J. Chen, Y. Yang, and H. Jin, "A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G," *Concurrency Computation Practice and Experience*, vol. 20, no. 15, pp. 1807–1820, 2008.

[15] Cloud exchange, 2011, http://cloudexchange.org.