An Efficient Checkpointing Scheme Using Price History of Spot Instances in Cloud Computing Environment

Daeyong Jung, Sungho Chin, Kwangsik Chung, Heonchang Yu, Joonmin Gil

An Efficient Checkpointing Scheme Using Price History of Spot Instances in Cloud Computing Environment

Daeyong Jung[1], SungHo Chin[1], KwangSik Chung[2], HeonChang Yu[1], JoonMin Gil[3*]

[1]Dept. of Computer Science and Education, Korea University, Seoul, Korea
[2]Dept. of Computer Science, Korea National Open University, Seoul, Korea
[3]School of Computer & Information Communications Engineering,
Catholic University of Daegu, Daegu, Korea
[1]{karat, wingtop, yuhc}@korea.ac.kr,
[2]kchung0825@knou.ac.kr, [3]jmgil@cu.ac.kr

Abstract. The cloud computing is a computing paradigm that users can rent computing resources from service providers as much as they require. A spot instance in cloud computing helps a user to utilize resources with less expensive cost, even if it is unreliable. When a user performs tasks with unreliable spot instances, failures inevitably lead to the delay of task completion time and cause a seriously deterioration in the QoS of users. Therefore, we propose a price history based checkpointing scheme based on SLA (Service Level Agreement) to avoid the delay of task completion time. The proposed checkpointing scheme reduces the number of checkpoint trials and improves the performance of task execution. The simulation results show that our scheme outperforms the existing checkpointing schemes in terms of the reduction of both the number of checkpoint trials and total costs per spot instances for user's bid.

Keywords: Cloud computing, Checkpointing, Spot instances, Price history

1 Introduction

Cloud computing is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers [1]. Typically, cloud computing services provide high level of scalability of IT resources with combined Internet technology to multiple customers [2].

Many definitions of cloud computing have been suggested [3, 4, 5]. Recently, several commercial cloud systems have been developed, such as Amazon EC2 [6], GoGrid [7], and FlexiScale [8]. Open-source cloud computing middlewares such as Eucalyptus [9], OpenNebula [10], and Nimbus [11] have been also provided in this literature. In the most of these clouds, the concept of an instance unit is used to pro-

* Corresponding author.

vide users with resources in a cost-efficient way. An instance means the VM (Virtual Machine) which is suitable for users' requirements. Generally, instances are classified into two types: on-demand instances and spot instances. The on-demand instances have a task execution for compute capacity by the hour with no long-term commitments. This frees users from the costs and complexities of planning, purchasing, and maintaining hardware and transforms what are commonly large fixed costs into much smaller variable costs [6]. While, the spot instances allow users to bid on unused cloud computing capacity and run those instances for as long as their bid exceeds the current spot price. The spot price changes periodically based on supply and demand, and users whose bids meet or exceed it gain access to the available spot instances. If users have flexibility in when applications can run, spot instances can significantly lower users' costs [6]. For task completion, therefore, spot instances have lower costs than on-demand instances. However, there is a problem that task failures can be occurred in the use of spot instances with higher cost than user suggested bid.

In this paper, we attempt to find a solution for an efficient checkpointing scheme in unreliable cloud computing environments and propose a price history based checkpointing scheme, by which users can pay the optimal cost based on SLA (Service Level Agreement). In the scheme, SLA management is done by the coordinator. The coordinator supports and manages SLA between users and instances. The failure of instances results in the delay of task completion time, so we design the cost-efficient checkpointing algorithm to solve the failure problem. In our proposed scheme, the checkpoints are taken on two points. One is the checkpoint taken on the rising edge in an execution bid when spot prices are more than a given threshold. The other is the checkpoint taken on the point when failure occurrence time is predicted by average execution time and failure possibility in an execution bid. Moreover, we carry out simulations to demonstrate the effectiveness of our scheme. Simulation results show that our scheme outperforms the existing schemes, such as hour-boundary checkpointing [17] and rising edge-driven checkpointing [14], in term of the reduction of both the number of checkpoint trials and total costs per spot instances for user' bid.

The rest of this paper is organized as follows: Section 2 briefly describes related work on checkpoint and SLA in cloud computing. Section 3 presents our system architecture and its components. Section 4 presents our SLA and checkpoint algorithms based on the price history of spot instances. Section 5 presents performance evaluations with simulations. Lastly, Section 6 concludes the paper.

2 Related Work

The unreliable cloud computing environment (spot instances) is less cost than reliable cloud computing environment (on-demand instances) in task processing environment. However, in unreliable cloud computing, it is difficult to estimate the total execution time of tasks and the total cost to be paid by users. Moreover, since task failures frequently occur according to the supply of instances and the demand of users on instances in unreliable cloud computing, many systems have used the checkpoint mechanisms to minimize task loss and reduce the rollback time of tasks.

In [12], authors proposed spot instance scheme that users can decide a minimum cost according to an SLA agreement between users and instances in Amazon's EC2. The scheme is based on a probabilistic model for the optimization of cost, performance and reliability and improved the reliability of service by changing dynamically conditions to satisfy user requirements. To improve the reliability of the services, this paper focuses on user costs rather than the point to be taken a checkpoint.

Due to the dynamic nature of the cloud computing, continuous monitoring on Quality of Service (QoS) attributes is necessary to enforce SLAs. In [13], authors proposed a mechanism for managing SLAs in cloud computing environment using the Web Service Level Agreement (WSLA) framework developed for SLA monitoring and SLA enforcement in a Service Oriented Architecture (SOA).

In [2], cloud platforms host several independent applications on a shared resource pool with the ability to allocate computing power to applications on per-demand basis. This paper proposed an autonomic resource manager to control the virtualized environment which decouples the provisioning of resources from the dynamic placement of virtual machines. This manager aims to optimize a global utility function which integrates both the degree of SLA fulfillment and computational costs.

In [13] and [2], authors focuses on cloud resource management in the reliable cloud computing environment, but this paper focuses on the unreliable cloud computing environment for the resource management applied to the SLA.

[14] introduced spot instances in the Amazon Elastic Compute Cloud (EC2) offer lower resource costs in exchange for reduced reliability. Based on the actual price history of EC2 spot instances, authors compared several adaptive checkpointing schemes in terms of monetary costs and the improvement of job completion time.

In this paper, we propose checkpoint scheme based on SLA to satisfy user requirements. Moreover, we compare our proposed checkpointing scheme with the existing checkpointing schemes (hour boundary checkpointing [17] and rising edge checkpointing [14]).

3 System Architecture

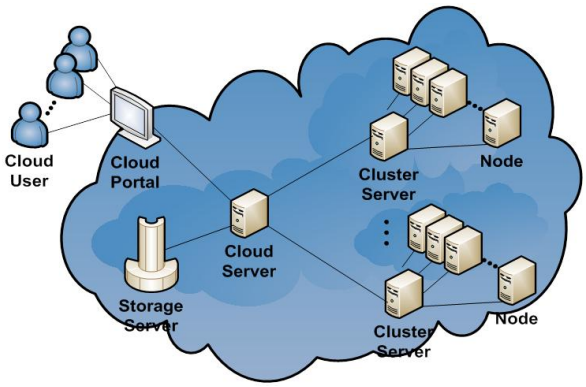

Fig. 1. Cloud computing environment

Fig. 1 shows the cloud computing environment assumed in this paper. This cloud computing environment basically consists of four entities: a cloud server, a storage server, cluster servers, and cloud users. The cloud server is connected to cluster servers and storage servers. The cluster server is composed of a lot of nodes. The cloud users can access the cloud server via the cloud portal to utilize the nodes in the cluster servers as resources. Therefore, the cloud server takes the responsibility of finding virtual resources to satisfy the user's requirements, such as SLA requirements and QoS requirements. The coordinator in the cloud server manages tasks and is responsible for the SLA management. We focus on the coordinator and the VM, which play an important role in our checkpointing scheme.

3.1 Layer Structure

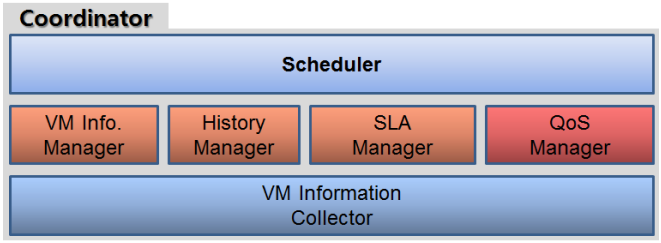

Fig. 2. The structure of Coordinator

Fig. 2 shows the structure of coordinator in the cloud server that is composed of scheduler, VM Information Manager, History Manager, SLA Manager, QoS Manager and VM Information Collector. In the coordinator, the four managers are responsible for generating and maintaining a list of available VMs, based on the information collected from VM Information Collector. The VM Information Collector collects VM information and provides it for VM information Manager. The VM Information Manager generates a list of CPU utilization, available memory and storage space, network bandwidth, and so on. The History Manager manages the history data, in which the past bid and execution time of spot instances are accumulated. SLA Manager and QoS Manager manage the SLA requirements and the QoS requirements, respectively. When a cloud user requests job execution, the Scheduler allocates the requested job to the selected VM.

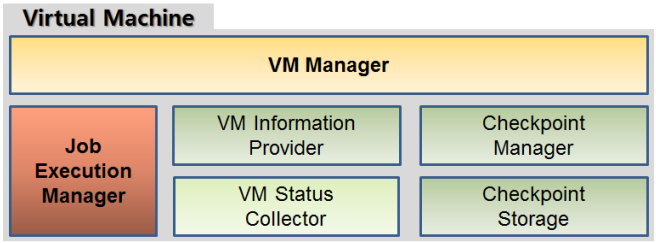

Fig. 3. The structure of Virtual Machine

Fig. 3 shows the structure of the VM. In this figure, VM Status Collector collects the status information of the VM, such as CPU utilization and memory space. VM Information Provider extracts resource information needed for job execution using the VM status Collector and delivers the resource information to VM Manager. Job execution Manager executes a requested job from the coordinator and returns a job result to VM Manager, and then VM Manager delivers the result to the coordinator. Checkpoint Manager manages checkpointing status and the data checkpointed by the Checkpoint Manager is stored to Checkpoint Storage.

3.2 Instances types

An instance means the VM that a cloud user uses. The instances are classified into two types: on-demand instances and spot instances. In on-demand instances, users can use VM resources after paying a fixed cost to lend instances per hour. On the other hand, using the spot instances, users can use VM resources only when the price of instances is smaller than other users' bid. The difference between the two instance types is as follows: in on-demand instances, a failure does not occur during task execution, but the cost is comparatively high. On the contrary, the cost of spot instances for task completion is lower than that of on-demand instances. However, task failures are inevitably encountered when there exist the instances with higher price than a user's bid.

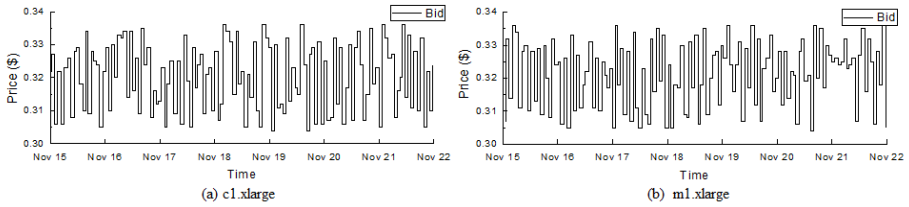

Fig. 4. Price history of EC2's spot instances

Amazon allows users to bid on unused EC2 capacity provided as 42 types of spot instances [15]. Their prices that are called *spot prices* are changed dynamically based on supply and demand. Fig. 4 shows examples of fluctuations of spot price for c1-xlarge (Standard Spot Instances - Extra Large) and m1-xlarge (High-Memory Spot Instances - Extra Large) during 7 days on November 2010 [16]. Our proposed system model is based on the characteristics of Amazon EC2's spot instances.

- The system provides a spot instance when user's bid is greater than the current price.
- The system stops immediately without any notice when user's bid is less than or equal to the current price. We call this an out-if-bid event or a failure.
- The system does not charge the latest partial hour when the system stops an instance.
- The system charges the latest partial hour when the user terminates an instance.

- The system provides the history of spot price.

4 The SLA based checkpointing scheme

In this section, we propose the SLA (Service Level Agreement) based checkpointing scheme in the spot instances.

4.1 SLA based on price history using spot instances

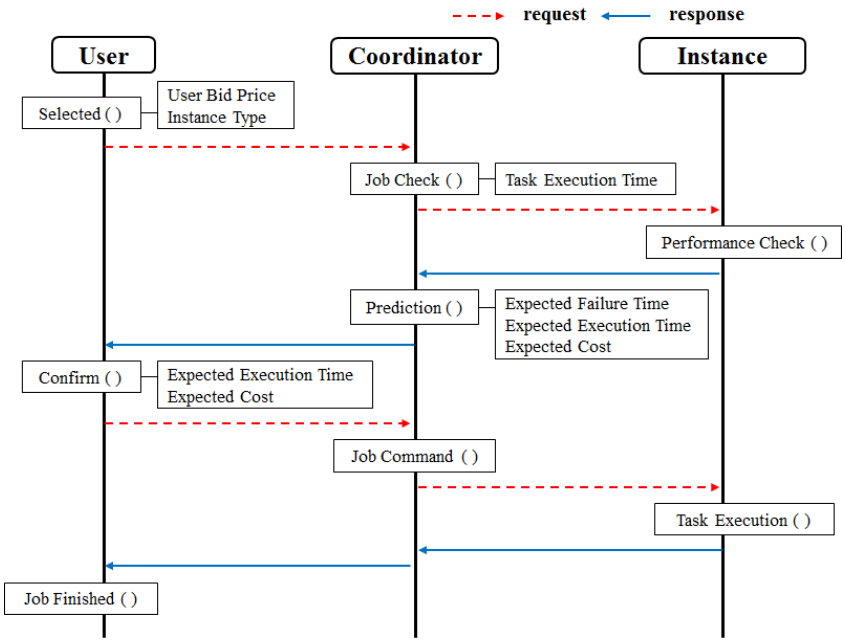

Fig. 5. SLA processing

Fig. 5 shows the process of SLA between a user and an instance. A user determines an instance type and the user's bid to begin tasks in the instance. The coordinator calculates a task execution time based on user configurations, such as the user's bid and the instance type. Then, the coordinator sends a request message to the selected instance to investigate the performance of the instance and calculates the expected execution time, the expected failure time and the expected cost. In addition, the coordinator sends a user the expected execution time and cost. When a task is completed in the selected instance, the coordinator receives task results from the instance and sends them to the user. In Fig. 5, the prediction function plays an important role in our SLA processing because it performs the estimation process of the expected failure time, the expected execution time, and the expected cost using price history. The following shows a detailed description for the prediction function.

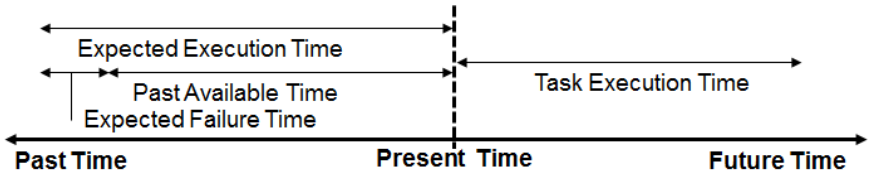

Fig. 6. Extraction of expected execution time from price history

Fig. 6 shows an illustrative example for task execution time, past available time, expected execution time, and expected failure time. The detailed definition for them is as follows:

- Task execution time: the total time needed to execute a task in the selected instance without failures.
- Past available time: the average execution time performed on the selected instance in the past time, excluding failure time. It is extracted from price history.
- Expected failure time: the time period when the spot price extracted from the price history exceeds a user's bid; *i.e.*, a total sum of failure time in the past time.
- Expected execution time: the sum of the past available time and the expected failure time.
- Total expected cost: the sum of costs that is charged for task execution.

4.2 Fault tolerance mechanisms using checkpoints

In the spot instance environment, a task fails when the cost exceeds the user's bid. Typically, this problem has been solved by using the checkpointing scheme, one of fault tolerance mechanisms [14]. In this section, we explain the existing checkpointing schemes and our proposed checkpointing scheme.

4.2.1 Hour-boundary checkpointing scheme

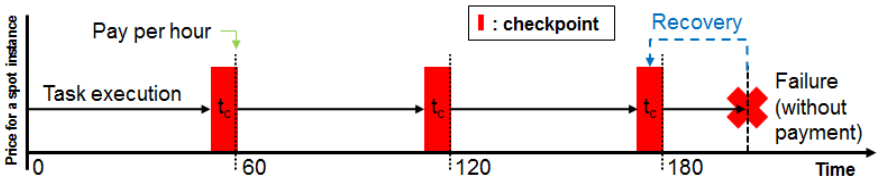

Fig. 7. Hour-boundary checkpointing

Fig. 7 illustrates the hour-boundary checkpointing scheme. This scheme takes a checkpoint in time boundaries, and a user pays the cost per hour without the user's bid. If the failure of a task is occurs, the running task is stopped. The task is restarted at the position of the last checkpoint.

4.2.2 Rising edge-driven checkpointing scheme

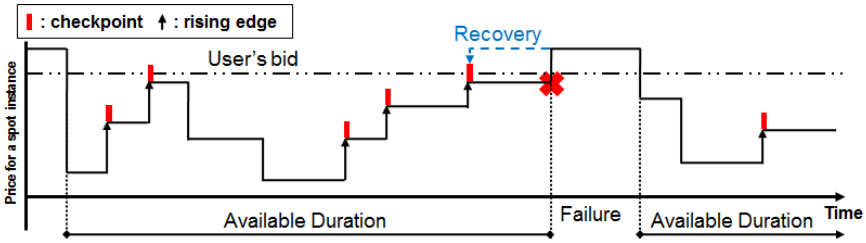

Fig. 8. Rising edge-driven checkpointing

Fig. 8 shows the rising edge-driven checkpointing scheme. This scheme takes a checkpoint when the cost is less than user's bid and the cost of spot instances is raised. It will increase the number of checkpoints significantly when cost is frequently fluctuated. The critical problem associated with this scheme is that the rollback time becomes long in case that the rising edge is not appeared in spot price for a long period after a checkpoint is taken. This leads to longer task completion time.

4.2.3 Our proposed checkpointing scheme

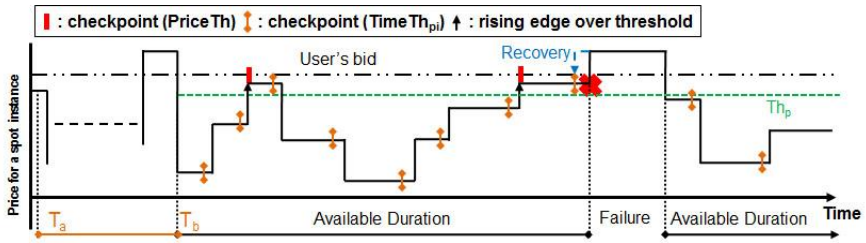

Fig. 9. Our proposed checkpointing scheme

Fig. 9 illustrates our proposed checkpointing scheme. This scheme basically performs checkpointing operation using two kinds of thresholds, price threshold and time threshold, based on the expected execution time of the price history. Now, let t_a and t_b denote, respectively, a start point and an end point in the expected execution time. Based on t_a and t_b, we obtain the price threshold ($PriceTh$) and the time threshold ($TimeTh_{p_i}$), which are used as thresholds in our proposed checkpoint scheme.

The price threshold, $PriceTh$, can be calculated by

$$PriceTh = \frac{P_{\min} + User_{bid}}{2}$$

where $User_{bid}$ represents the bid suggested by the user. $P_{\min}$ represents an available minimum price in a period between t_a and t_b as follows:

$$P_{\min} = PriceMin(t_a, t_b)$$

The time threshold of price P_i, $TimeTh_{p_i}$, can be calculated by

$$TimeTh_{p_i} = AvgTime_{P_i}(t_a, t_b) \times (1 - F_{p_i})$$

where F_{p_i} is the failure probability of price P_i and $AvgTime_{P_i}(t_a, t_b)$ represents the average execution time of P_i in a period between t_a and t_b.

```
1:  Boolean flag = false          // a flag representing occurrence of a task failure
2:  while (!task execution finishes) do
3:    if (spot prices < User's bid ) then
4:      if (flag) then
5:        Recovery ( );
6:        flag = false;
7:      end if
8:      if (!flag) then
9:        if (rising edge && Price Threshold ≤ spot prices) then
10:         Checkpoint ( );
11:       end if
12:       if (Time Threshold < execution time in current price) then
13:         Checkpoint ( );
14:       end if
15:     end if
16:   end if
17:   if (failure is occurred) then
18:     flag = true;
19:   end if
20: end while
21: Function Checkpoint ( )
22:   take a checkpoint on the spot instance;
23:   send the checkpoint to the storage;
24: end Function
25: Function Recovery ( )
26:   rollback the checkpoint to the storage;
27:   restart the job execution;
28: end Function
```

Fig. 10. Checkpointing and recovery algorithms

Using these two thresholds, our proposed checkpointing scheme performs checkpoint operations according to two cases: first case is that a checkpoint is performed when there is a rising edge between the user's bid and the price threshold. Second case is based on the failure probability and average execution time of each price. A checkpoint is performed when the time threshold exceeds the execution time of current price.

Fig. 10 shows the checkpointing and recovery algorithms used in our proposed scheme. In the algorithms, the flag representing the occurrence of a task failure is

initially set to false. The checkpointing process repeats until all tasks are completed. When task execution is normal (i.e., the flag is false), the scheduler performs checkpoint process to provide against job failure (lines 2-20). Recovery process is performed when the flag is true (lines 4-7). Two cases of checkpoints are performed (lines 8-15). If the rising spot price is between user's bid and price threshold, the scheduler performs checkpointing operation (lines 9-11). If the execution time is greater than the time threshold, the scheduler also performs checkpointing operation (lines 12-14). When task failure event occurs, the flag is set to true to invoke the recovery function (lines 17-19). Lines 21-24 and 25-28 show a detail process of the checkpointing and recovery, respectively.

Our proposed scheme can reduce checkpointing overhead because the number of checkpoints is less than that of the existing checkpointing schemes (hour-boundary checkpointing and rising edge-driven checkpointing). In our proposed scheme, the two thresholds and expected failure time are calculated based on price history. The thresholds are dynamically changed according to the price behavior of instances in the price history.

5 Performance Evaluation

In this section, we evaluate the performance of our checkpointing scheme using simulations and compare it with that of the other checkpointing schemes.

5.1 Simulation Environments

Our simulations are conducted using the history data obtained from the Amazon EC2's spot instances [16], which is accumulated during a period from 11-15-2010 to 11-22-2010 as shown in Fig. 4. The history data before 11-18-2010 are used to extract the expected execution time and failure occurrence probability for our checkpointing scheme. The applicability of our checkpointing scheme is tested using the history data after 11-18-2010, which are also used for hour-boundary checkpointing and rising edge-driven checkpointing schemes.

In the simulations, two types of spot instances are applied to show the effect of two different resource types on the performance of three checkpointing schemes; one resource type is a computing-type instance and another type is a memory-type instance. Table 1 shows the resource types used in our simulation. In this table, c1.xlarge offers more compute units than other resources and can be used for compute-intensive applications. On the other hand, m1.xlarge offers much 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 checkpointing scheme with that of the two checkpointing schemes in terms of the task execution time, the failure time, the number of failures, and the number of checkpoints.

Table 1. Resource types

Instance type name	Compute unit	Virtual cores	Memory	Storage	Platform
c1.xlarge (computing Instance)	8 EC2	4core (2 EC2)	15GB	1690GB	64-bit
m1.xlarge (high-memory Instance)	6.5 EC2	2core (3.25 EC2)	17.1GB	420GB	64-bit

5.2 The analysis of computing-type instances

Before analyzing the performance of our checkpointing scheme, we firstly extract parameter values from the spot history presented in Fig. 4(a). Table 2 shows the simulation parameters and values used for the analysis of computing-type instances.

Table 2. Simulation parameters and values for c1.xlarge instance

Simulation parameter	Task time	Max Bid	Average bid	Min bid	Checkpoint time	Recovery time
Value	259200(s)	0.336($)	0.319($)	0.304($)	300(s)	300(s)

We also extract the failure occurrence probability for each price from the spot history (11-15-2010 ~ 11-18-2010) presented in Fig. 4(a). The extracted failure occurrence probability is used to determine the time threshold in our checkpointing scheme. Fig. 11 shows the failure occurrence probability for c1.xlarge instance. In this figure, X and Y-axis mean spot price and failure occurrence probability per spot price for a given user's bid, respectively.

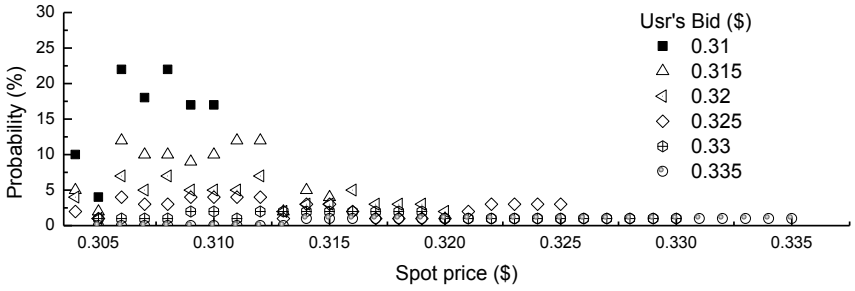

Fig. 11. Failure occurrence probability for c1.xlarge instance

Fig. 12 shows the performance comparison of our checkpointing scheme with hour-boundary checkpointing and rising edge-driven checkpointing schemes when tasks in c1.xlarge instance are used. Fig. 12(a) shows the effect of total task execution time and total failure time on the performance of three checkpointing schemes. Fig. 12(b) shows the effect of the number of failures and checkpoints in each user's bid on the performance of three checkpointing schemes.

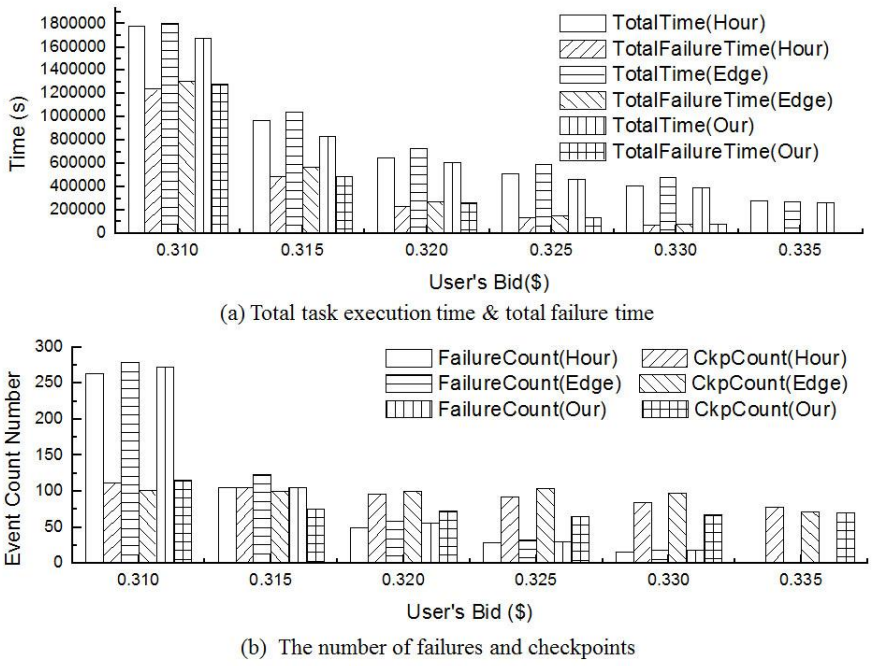

Fig. 12. Performance comparison of checkpointing schemes in c1.xlarge

From this figure, we can find that our checkpointing scheme achieves performance improvements in an average task execution time of 7.9% over the hour-boundary checkpointing scheme and in an average task execution time of 14.3% over the rising edge-driven checkpointing scheme. We can also find that our scheme reduces the number of checkpoints by average of 17 times over the hour-boundary checkpointing scheme and by average of 18 times over the rising edge-driven checkpointing scheme.

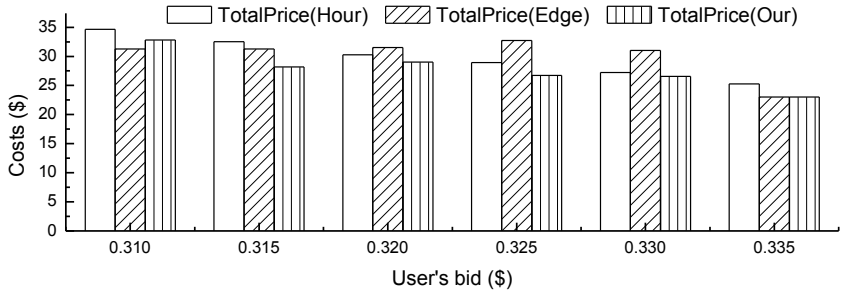

Fig. 13. Comparison of total costs in c1.xlarge

Fig. 13 shows the total costs in each user's bid. From this figure, we can see that our checkpointing scheme reduces the costs by average of $2.08 over the hour-

boundary checkpointing scheme and by average of $2.42 over the rising edge-driven checkpointing scheme.

5.3 The analysis of memory-type instances

Now, we present the performance evaluation of our checkpointing scheme when memory-type instances are used. As the analysis presented in previous subsection, we firstly extract parameter values from the spot history presented in Fig. 4(b). Table 3 shows the simulation parameters and values used for the analysis of memory-type instances.

Table 3. Simulation value of m1.xlarge instance

Simulation parameter	Task time	Max bid	Average bid	Min bid	Checkpoint time	Recovery time
Value	259200(s)	0.76($)	0.32($)	0.304($)	300(s)	300(s)

We also extract the failure occurrence probability for each price from the spot history (11-15-2010 ~ 11-18-2010) in Fig. 4(b). Fig. 14 shows the failure occurrence probability for m1.xlarge instance. In this figure, X and Y-axis mean spot price and failure occurrence probability per spot price for a given user's bid, respectively.

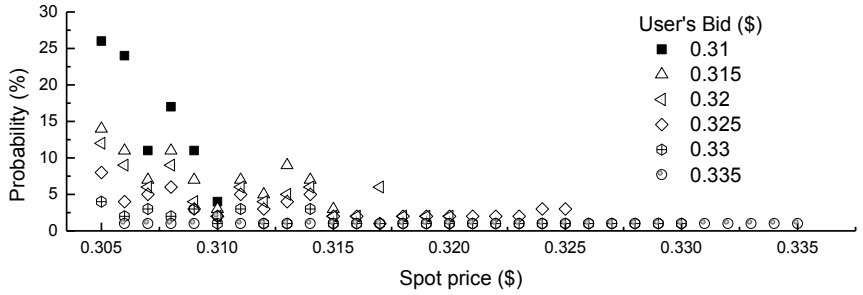

Fig. 14. Comparison of fault occurrence probability in m1.xlarge

Fig. 15 shows the performance comparison of our checkpointing scheme with hour-boundary checkpointing and rising edge-driven checkpointing schemes when tasks in the m1.xlarge instance are used. Fig. 15(a) shows the effect of total task execution time and total failure time on the performance of three checkpointing schemes. Fig. 15(b) shows the effect of the number of failures and checkpoints in each user's bid on the performance of three checkpointing schemes.

From this figure, we can find that our checkpointing scheme achieves performance improvements in an average task execution time of 14.35% over the hour-boundary checkpointing scheme and in an average task execution time of 23.83% over the rising edge-driven checkpointing scheme. We can also find that our scheme reduces the number of checkpoints by average of 28 times over the hour-boundary checkpointing scheme and by average of 31 times over the rising edge-driven checkpointing scheme.

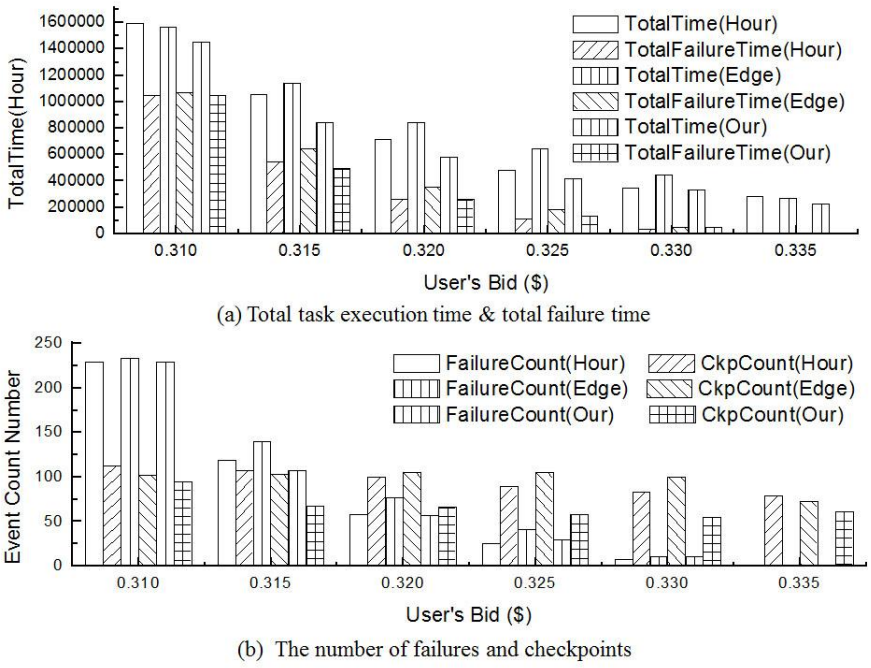

Fig. 15. Performance comparison of checkpointing schemes in m1.xlarge

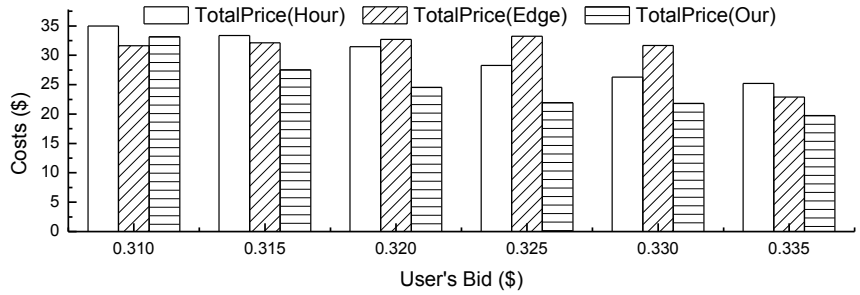

Fig. 16. Comparison of total costs in m1.xlarge

Fig. 16 shows the total costs in each user's bid. From this figure, we can see that our checkpointing scheme reduces the costs by average of $5.15 over the hour-boundary checkpointing scheme, and by average of $5.93 over the rising edge-driven checkpointing scheme.

6 Conclusion

In this paper, we proposed an efficient checkpointing scheme using the price history of spot instances to improve the stability of task processing in unreliable cloud

computing environment. Our proposed scheme basically performs checkpointing operation based on two kinds of thresholds, price threshold and time threshold. These two thresholds were extracted from the price history of spot instances and used to determine checkpointing position in cost-efficient way in the presence of the failures of spot instances arisen from price fluctuation. As a result, our scheme can significantly reduce the number of checkpoint trials compared to the existing checkpointing schemes. Furthermore, the rollback time of our scheme can be much lesser than that of the existing checkpointing schemes because our scheme can adaptively perform checkpointing operation according to the time and price of spot instances. Simulation results showed that our scheme can achieve cost efficiency by reducing rollback time per instance for a given user's bid regardless of the resource types of spot instaces. In the future, we have a plan to expand our environment into a combination of spot instances and on-demand instances for various cloud computing services.

Acknowledgments. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. 2010-0015637).

References

[1] Buyya, R., Chee Shin, Y., Venugopal, S.: Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: Proceeding of the 10th IEEE International Conference on High Performance Computing and Communications, pp. 5-13. (2008)

[2] Van, H.N., Tran, F.D., Menaud, J.-M.: SLA-Aware Virtual Resource Management for Cloud Infrastructures. In: Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology, vol. 2, pp. 357-362. IEEE Computer Society (2009)

[3] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. EECS Department, University of California, Berkeley (2009)

[4] Youseff, L., Butrico, M., Da Silva, D.: Toward a Unified Ontology of Cloud Computing. In: Grid Computing Environments Workshop, 2008. GCE '08, pp. 1-10. (2008)

[5] Foster, I., Yong, Z., Raicu, I., Lu, S.: Cloud Computing and Grid Computing 360-Degree Compared. In: Grid Computing Environments Workshop, 2008. GCE '08, pp. 1-10. (2008)

[6] Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2 (2011)

[7] GoGrid, http://www.gogrid.com, (2011)

[8] FlexiScale, http://www.flexiscale.com, (2011)

[9] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus Open-Source Cloud-Computing System. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 124-131. IEEE Computer Society (2009)

[10] OpenNebula, http://www.opennebula.org, (2011)

[11] Nimbus, http://workspace.globus.org, (2011)

[12] Andrzejak, A., Kondo, D., Yi, S.: Decision Model for Cloud Computing under SLA Constraints. In: Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis

and Simulation of Computer and Telecommunication Systems, pp. 257-266. IEEE Computer Society (2010)
[13] Patel, P., Ranabahu, A., Sheth, A.: Service Level Agreement in Cloud Computing. In: Proceedings of Conference on Object Oriented Programming Systems Languages and Applications, pp. 212-217. (2009)
[14] Yi, S., Kondo, D., Andrzejak, A.: Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, pp. 236-243. IEEE Computer Society (2010)
[15] Amazon EC2 spot Instances, http://aws.amazon.com/ec2/spot-instances/, (2010)
[16] Cloud exchange, http://cloudexchange.org, (2011)
[17] Yi, S., Heo, J., Cho, Y., Hong, J.: 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)