Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 217547, 12 pages
http://dx.doi.org/10.1155/2014/217547

Research Article

Estimated Interval-Based Checkpointing (EIC) on Spot Instances in Cloud Computing

Daeyong Jung, JongBeom Lim, Heonchang Yu, and Taeweon Suh

Department of Computer Science Education, Korea University, Seoul, Republic of Korea

Correspondence should be addressed to Taeweon Suh; suhtw@korea.ac.kr

Received 21 January 2014; Accepted 6 May 2014; Published 28 May 2014

Academic Editor: Young-Sik Jeong

In cloud computing, users can rent computing resources from service providers according to their demand. Spot instances are unreliable resources provided by cloud computing services at low monetary cost. When users perform tasks on spot instances, there is an inevitable risk of failures that causes the delay of task execution time, resulting in a serious deterioration of quality of service (QoS). To deal with the problem on spot instances, we propose an estimated interval-based checkpointing (EIC) using weighted moving average. Our scheme sets the thresholds of price and execution time based on history. Whenever the actual price and the execution time cross over the thresholds, the system saves the state of spot instances. The Bollinger Bands is adopted to inform the ranges of estimated cost and execution time for user's discretion. The simulation results reveal that, compared to the HBC and REC, the EIC reduces the number of checkpoints and the rollback time. Consequently, the task execution time is decreased with EIC by HBC and REC. The EIC also provides the benefit of the cost reduction by HBC and REC, on average. We also found that the actual cost and execution time fall within the estimated ranges suggested by the Bollinger Bands.

1. Introduction

Cloud computing is a computing paradigm that constitutes an advanced computing environment that evolved from utility and grid computing. The infrastructure of cloud computing typically includes a collection of interconnected and virtualized computers from parallel and distributed systems. The virtual computers are dynamically provided to consumers as one or more unified computing resources, based on service level agreements (SLA) established through negotiation between the service provider and consumers [1–5]. Typically, cloud computing services provide a high level of scalability of computing resources combined with Internet technology to multiple customers [6]. Currently, there are several commercial cloud systems in service such as Amazon EC2 [7], GoGrid [8], and FlexiScale [9].

In most of these cloud services, there is a notion of an instance to provide users with resources in a cost-efficient manner. An instance means the virtual machine (VM) that serves for the user's need. In general, instances are classified into two types: on-demand instances and spot instances. On-demand instances are charged for the compute capacity on an hourly basis without the long-term commitment. This frees users from the costs and complexities of planning, purchasing, and maintaining hardware and transforms commonly large fixed costs into much smaller variable costs [7]. On the other hand, spot instances allow users to bid on unused compute capacity and utilize those instances for as long as the current spot price is below their bid. The spot price is changing periodically based on supply and demand. When users' bids meet or exceed the price, they gain access to the available spot instances. If users are flexible as to when applications should run, spot instances can significantly decrease the cost as reported in [7]. Nevertheless, there is a risk of task failures, which occurs when the spot price of the instance becomes higher than the bid price.

To efficiently handle this problem, the checkpointing schemes have been proposed in the research community [10, 11]. The checkpointing saves the execution status of tasks if a certain condition is met and then recovers the task status from the last saved point upon a failure. It allows a reduction in the execution time and cost in an unreliable computing environment. On a legal side, the SLA is typically used for alleviating the uncertainty by specifying service details such as price and task execution time. SLA specifies the resource allocation and rental terms to consumers in agreement with providers.

In this paper, we propose the estimated interval-based checkpointing (EIC), which improves the efficiency over our previous study [12]. The key idea is adopting the weighted moving average (WMA) and Bollinger Bands. The moving average is a history-based prediction scheme. The WMA sets a different weight for each time interval in the past and calculates the average of the weights. With these weights, the failure occurrence probability is obtained in each interval. The threshold for checkpointing is calculated based on the average failure probability. We apply two thresholds of price and time in EIC. In addition, we use the Bollinger Bands to inform users of estimated execution time and cost. In the stock market, the Bollinger Bands is a well-known analysis method. It is used to measure the high and low value level of the previous trading data. This method is used to predict the price bid in the stock market. We use the Bollinger Bands to calculate both the estimated execution time and the cost.

We have measured the number of checkpoint trials and total cost per spot instance for a user bid. Simulation results show that the EIC outperforms the existing schemes, hour-boundary checkpointing (HBC) [13] and rising edge-driven checkpointing (REC), [11] in terms of the number of checkpoints. Consequently, the EIC minimizes the execution time of applications and the time wasted by task failures.

The rest of this paper is organized as follows. Section 2 briefly describes related work on resource allocation, SLA, fault tolerance, moving average, and Bollinger Bands in cloud computing. Section 3 presents our system architecture. Section 4 presents our SLA, estimation, and checkpoint algorithms based on the price history of spot instances. Section 5 presents performance evaluations with simulations. Finally, Section 6 concludes the paper.

2. Related Work

Many researchers and companies have recently studied fault-tolerance techniques in two different environments of cloud computing: reliable environments, with on-demand instances [14, 15], and unreliable environments, with spot instances [11, 13, 16, 17]. The fault-tolerance techniques are more required in unreliable environments. Our study was performed in the latter category of the environments to provide the cost-effectiveness of task execution.

Spot instances are typically used in unreliable environments, and studies on spot instances focus on performing tasks at low monetary costs. The spot instances in the Amazon Elastic Compute Cloud (EC2) offer lower price at the expense of the reduced reliability [18]. Cloud exchange [19] supports the actual price history of EC2 spot instances. In the spot instances environment, there are numerous studies on resource allocation [16, 17], SLA [6, 20, 21], fault tolerance [10, 11, 13, 16], moving average [22, 23], and Bollinger Bands [24, 25].

On the resource allocation side, Voorsluys and Buyya [16] solve the problem of running computation-intensive tasks on a pool of intermittent VMs. To mitigate potential unavailability periods, the study proposed a multifaceted fault-aware resource provisioning policy. Their solution employs price and runtime estimation mechanisms. The proposed strategy achieves cost savings and stricter adherence to deadlines. Zhang et al. [17] introduced a solution of how best to match customer demand in terms of both supply and price and to maximize the provider's revenue and the customer's satisfaction in terms of VM scheduling. The proposed model is designed to solve the problem of discrete-time optimal control. This model achieves higher revenues than static allocation strategies and minimizes the average request waiting time. Our work differs from [16, 17] in that we focus on reducing the rollback time after a task failure, achieving the cost savings and reducing the total execution time.

On the SLA side, Andrzejak et al. [20] proposed a probabilistic decision model to help users decide a minimum cost according to an SLA between users and Amazon's EC2. The scheme is based on a probabilistic model for the optimization of cost, performance, and reliability. It improves the reliability of service by changing conditions dynamically to satisfy user requirements. Due to the dynamic nature of cloud computing, continuous monitoring of the quality of service (QoS) attributes is necessary to enforce SLAs. Two similar studies [6, 21] focus on cloud resource management in the reliable cloud environment. One is based on SLA monitoring and enforcement in a service-oriented architecture (SOA) [21], whereas the other focuses more on the resource management. The resource manager optimizes a global utility function that integrates both the SLA fulfillment degree and the computational costs [6]. Our paper differs from [6, 21] in that we deal with the resource management in the unreliable cloud environment.

On the fault tolerance side, two similar studies (HBC [13] and REC [11]) proposed enforcing fault tolerance in cloud computing with spot instances. 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. Goiri et al. [10] evaluated three fault tolerance schemes, checkpointing, migration, and job duplication, assuming that the communication cost is fixed. The migration-based scheme shows a better performance than the checkpointing or the job duplication-based scheme. Voorsluys and Buyya [16] also analyzed and evaluated the impact of checkpointing and migration on fault tolerance using spot instances. Our paper differs from [10, 11, 13, 16] in that we utilize double thresholds for fault tolerance.

On the moving average and Bollinger Bands side, the moving average takes the next observation data using the data in the past [22, 23]. Reference [22] introduced the simple moving average (SMA) and WMA. Reference [23] used the average data to apply weight according to each interval. It evaluates the average of price depending on the weight change. Our paper also adopts WMA to estimate price, execution time, and thresholds based on price history. However, we found that the estimation is not accurate enough. We overcome this shortcoming by applying Bollinger Bands to estimate the execution time and the price ranges. The Bollinger Bands, proposed by Bollinger [24], is a technical

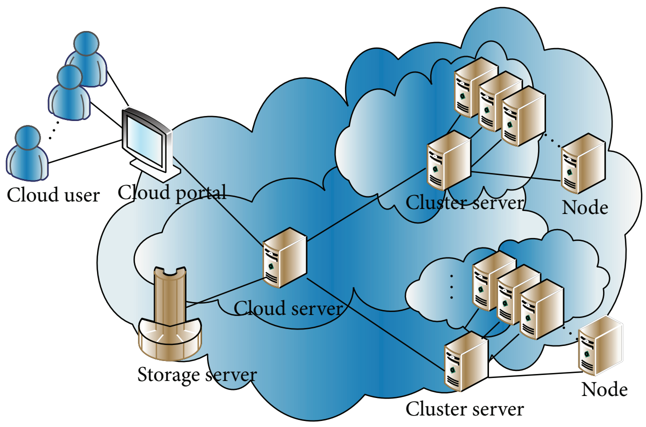


Figure 1: Cloud computing environment.

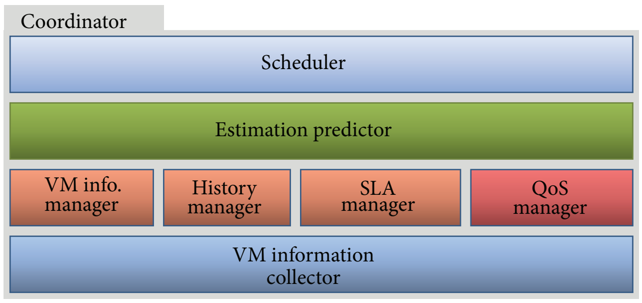


Figure 2: Cloud computing environment.

analysis method used in the stock market. It analyzes previous trades and determines the standard deviation. Daytrader [25] introduced a method to predict the range of Bollinger Bands. This prediction requires the selection of length of the moving average around which the Bollinger Bands are plotted, and standard deviations to calculate from this moving average. Our paper differs from [24, 25] in that we apply Bollinger Bands to predict both cost and execution time ranges in the unreliable cloud environment.

In our previous paper [12], we proposed a checkpoint scheme based on SLA to satisfy user requirements. Our previous study performs a checkpointing operation based on two thresholds: price and time. The estimated execution time is predicted using the price history of an instance only for the same amount of time in task execution in the past. This paper differs in that the Bollinger Bands was adopted to improve the accuracy of cost and execution time estimations with utilizing numerous estimation intervals of the past.

3. System Architecture

Figure 1 shows the cloud computing environment assumed in this paper, which basically consists of four entities: a cloud server, storage servers, cluster servers, and cloud users. The cloud server is connected to cluster servers and storage servers. The cluster server is composed of many nodes. 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 responsibility of finding resources and spawning virtual machines to satisfy the user's requirements in terms of the SLA and QoS. 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 important roles in our checkpointing scheme.

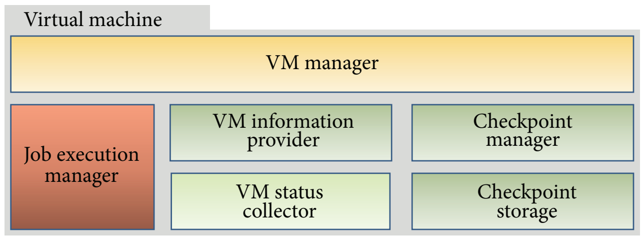


Figure 3: The structure of virtual machine.

3.1. Layer Structure. Figure 2 shows the structure of the coordinator in the cloud server, which is composed of Scheduler, Estimation Predictor, 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 to 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. Estimation Predictor analyzes data taken from the other managers and calculates the range of estimation completion time and total prices. When a cloud user requests job execution, the Scheduler allocates the requested job to the selected VM.

Figure 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 received from the coordinator and returns a job result to VM Manager, and VM Manager then delivers the result to the coordinator. Checkpoint Manager manages checkpointing status and the data checkpointed by the Checkpoint Manager are stored in Checkpoint Storage.

3.2. Instances Types. 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. In contrast, the cost of spot instances is lower than that of on-demand instances. However, there is an inevitable risk of task failures encountered when the price of the instances becomes higher than the user bid.

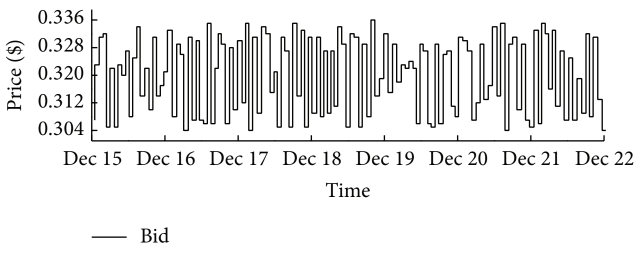


FIGURE 4: Price history of EC2's spot instances for c1-xlarge.

Amazon allows users to bid on unused EC2 capacity among 42 types of spot instances [18]. Their prices, which are referred to as spot prices, are changing dynamically based on supply and demand. Figure 4 shows a spot price fluctuations example during seven days in December 2010 for c1-xlarge (High-CPU Spot Instances—Extra Large) [19]. Our proposed system model is based on the characteristics of Amazon EC2's spot instances.

(i) The system provides a spot instance when a user bid is higher than the current price.

(ii) The system immediately stops the spot instance without any notice when a user bid becomes less than or equal to the current price. We refer to it as an out-of-bid event or a failure.

(iii) The system does not charge for the last partial hour when the system stops the spot instance.

(iv) The system does charges for the last partial hour when the user voluntarily terminates the spot instance.

(v) The system provides the spot price history.

4. Estimated Interval-Based Checkpointing

In this section, we detail an estimated interval-based checkpointing for spot instances that includes the SLA, the moving average, Bollinger Bands, and the fault tolerance.

4.1. Price History-Based SLA. Figure 5 shows the SLA process between a user and an instance. A user determines an instance type and the bid price to begin tasks on the instance. The coordinator calculates the task execution time based on the user's decision. 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. Then, the coordinator sends a user the expected execution time and cost. When a task is completed on the selected instance, the coordinator receives the outcome from the instance and sends it to the user. As shown in Figure 5, the prediction function in the coordinator plays an important role in our SLA process because it performs the estimation using price history.

4.2. Estimation Using Moving Average and Bollinger Bands. In EIC, the checkpointing operation is performed by analyzing the price variation at certain time intervals in the past. We use the moving average which estimates a job execution time and a cost from the analyzed data. The estimations are combined with the failure probability to calculate the thresholds for the checkpointing operation. The proper estimation of the execution time and cost is crucial for the credibility of service providers to customers. For the probable estimation information, we use Bollinger Bands. It suggests the upper and lower bounds of the execution time and the cost. We show in Section 5 that the actual execution time and the cost fall within the bounds.

In this paper, we introduce a terminology referred to as estimated interval (EI). Figure 6 shows an illustrative definition of the EI. The detailed definitions are as follows.

(i) Pure task time: the time to execute a task on a selected instance when there are no failures.

(ii) Past pure task time: a sum of time durations taken for task execution on the selected instance in the past, excluding failure durations. It is extracted from the price history.

(iii) Past failure time: a sum of failure durations in the past to execute a task. A failure occurs when the current user bid is below the past spot price.

(iv) Estimated interval (EI): the sum of the past pure task time and the past failure time.

(v) Moving average EI: the average of EIs computed using moving average.

(vi) Expected cost: the average of costs charged for task execution in EIs.

Based on the simple moving average (SMA), we calculate an estimated time $\mathrm{SMA}_{\mathrm{ET}}$ and an estimated price $\mathrm{SMA}_{\mathrm{Ep}}$ by the average of EIs in the price history, as shown below:

$$\mathrm{SMA}_{\mathrm{ET}}(n) = \frac{\mathrm{ET}_1 + \mathrm{ET}_2 + \mathrm{ET}_3 + \cdots + \mathrm{ET}_n}{n},$$
$$\mathrm{SMA}_{\mathrm{EP}}(n) = \frac{\mathrm{EP}_1 + \mathrm{EP}_2 + \mathrm{EP}_3 + \cdots + \mathrm{EP}_n}{n}, \tag{1}$$

where ET_i is the estimated time in an interval i, EP_i is the estimated price in an interval i, and n is the number of intervals, as depicted in Figure 6.

Based on the weighted moving average (WMA), $\mathrm{WMA}_{\mathrm{ET}}$ and $\mathrm{WMA}_{\mathrm{EP}}$ are averages of the estimated time and the estimated price from the price history with a weight using SMA. They are calculated by

$$\mathrm{WMA}_{\mathrm{ET}} = \frac{\sum_{i=1}^{n} \alpha_i \mathrm{ET}_{n-i+1}}{\sum_{i=1}^{n} \alpha_i}, \qquad \mathrm{WMA}_{\mathrm{EP}} = \frac{\sum_{i=1}^{n} \alpha_i \mathrm{EP}_{n-i+1}}{\sum_{i=1}^{n} \alpha_i}, \tag{2}$$

where α is a weight. The α_i is assigned the highest for the most recent EI_1, and it is decreased from the most recent EI_1 to the last EI_N. The weight α_i is calculated by

$$\alpha_i = \frac{n + 1 - i}{\sum_{i=1}^{n} i}, \tag{3}$$

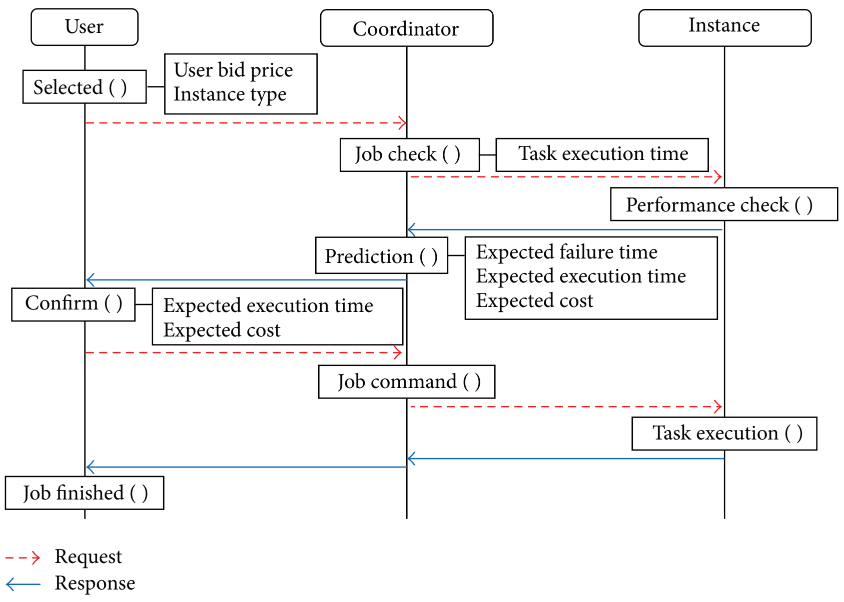
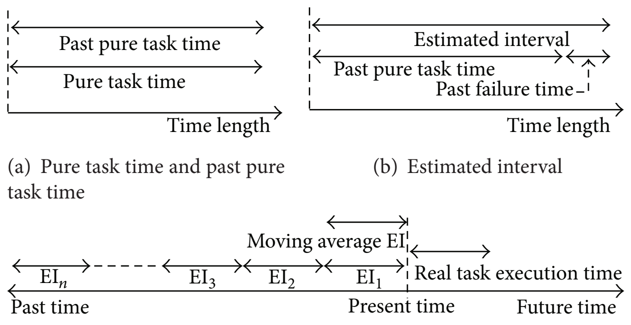


FIGURE 5: SLA processing.



(a) Pure task time and past pure task time
(b) Estimated interval
(c) Moving average estimated interval

FIGURE 6: Moving average relation.

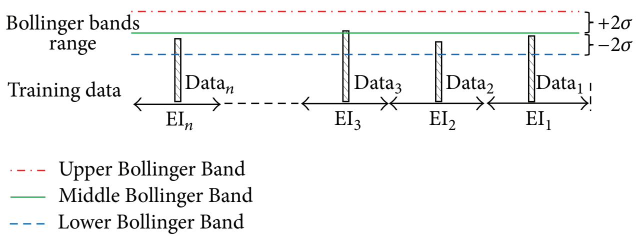


FIGURE 7: Bollinger Bands acquisition.

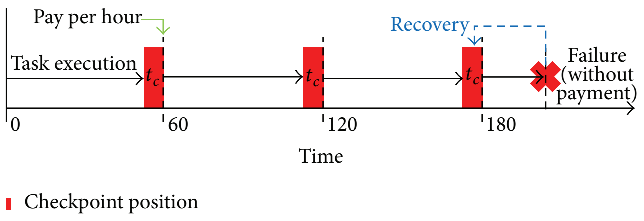


FIGURE 8: Hour-boundary checkpointing.

where i and n are the interval number and the last interval number, respectively. By adjusting the weight, we empirically reduce the gap between the estimation and actual data from real execution. The Bollinger Bands presents the range of estimation using a moving average and a standard deviation. Generally, the Bollinger Bands itself adopts a moving average as the middle value. We use WMA as the middle value of the Bollinger Bands because the near past is more likely to be influencing the near future. The upper and lower bounds of the Bollinger Bands are defined as

(i) Middle Bollinger Band = WMA

(ii) Lower Bollinger Band = Middle Bollinger Band $- 2\sigma$

(iii) Upper Bollinger Band = Middle Bollinger Band $+ 2\sigma$

where σ is the standard deviation of EIs. Figure 7 illustrates the range of Bollinger Bands using training data that consist of each estimation value in EIs. The training data are obtained from (an) N-zone EIs.

4.3. Fault Tolerance Mechanisms Using Checkpoints. On a spot instance, a task failure occurs when the user's bid is below the current spot price. This problem has been solved by using the checkpointing, one of fault tolerance mechanism [9]. In this section, we detail the existing checkpointing methods and our proposed scheme.

Figure 8 illustrates the hour-boundary checkpointing (HBC). In this scheme, the checkpointing operation is performed in the hour boundary, and a user pays the biding price on an hourly basis. Upon the task failure, the task is restarted from the position of the last checkpoint.

Figure 9 illustrates the rising edge-driven checkpointing (REC). In this scheme, the checkpointing operation is performed when both the price of the spot instance is raised (i.e.,

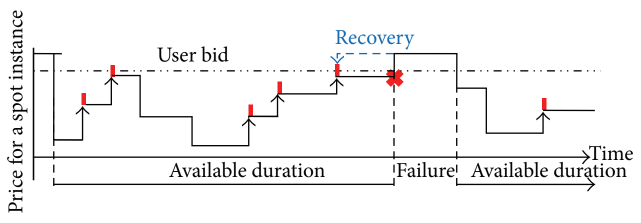


FIGURE 9: Rising edge-driven checkpointing.

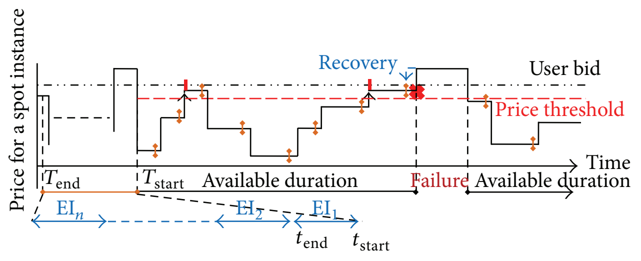


FIGURE 10: Estimated interval-based checkpointing.

rising-edge) and the price is less than the user bid. It increases the number of checkpoints when the spot price fluctuates frequently. The critical problem in REC is that the rollback time becomes long when a rising edge does not appear for a long period of time after a checkpoint is taken. This could lead to a longer time for the task completion than HBC.

Figure 10 illustrates checkpointing operation in EIC. It is basically performed using two thresholds, price and time, based on the expected execution time according to the price history. Now, let t_{start} and t_{end} denote a start point and an end point, respectively, in the total of EIs. Based on t_{start} and t_{end}, we obtain the price threshold (PriceTh) and the time threshold (TimeTh_{P_i}), which are used as thresholds in EIC.

The price threshold, PriceTh, can be calculated by

$$\text{PriceTh} = \frac{\text{WP}_{\min} + \text{User}_{\text{bid}}}{2}, \tag{4}$$

where User_{bid} represents the user bid and $\text{WP}_{\min}$ represents an available minimum price using a moving average in the time duration between t_{start} and t_{end}.

First, the $P_{\min}^{\text{EI}_i}$ represents the minimum price in the time duration between T_{start} and T_{end} in EI_i

$$P_{\min}^{\text{EI}_i} = \text{PriceMin}\left(t_{\text{start}}, t_{\text{end}}\right). \tag{5}$$

Second, the $\text{WP}_{\min}$ is the average of the product of $P_{\min}^{\text{EI}_i}$ and sum of the weighted value α_i:

$$\text{WP}_{\min} = \frac{\sum_{i=1}^{n} \alpha_i \times P_{\min}^{\text{EI}_{n-i+1}}}{\sum_{i=1}^{n} \alpha_i}. \tag{6}$$

The time threshold of price P_i, TimeTh_{P_i}, is calculated by

$$\text{TimeTh}_{P_i} = \frac{\sum_{j=1}^{n} \alpha_j \times \text{TimeTh}_{P_i}^{\text{EI}_{n-j+1}}}{\sum_{j=1}^{n} \alpha_j}. \tag{7}$$

In each EI, the time threshold of price P_i, $\text{TimeTh}_{P_i}^{\text{EI}_j}$, is calculated by

$$\text{TimeTh}_{P_i}^{\text{EI}_j} = \text{AvgTime}_{P_i}^{\text{EI}_j}\left(t_{\text{start}}, t_{\text{end}}\right) \times \left(1 - F_{P_i}^{\text{EI}_j}\right), \tag{8}$$

where $F_{P_i}^{\text{EI}_j}$ is the failure probability of price P_i and $\text{AvgTime}_{P_i}^{\text{EI}_j}(t_{\text{start}}, t_{\text{end}})$ represents the average execution time of P_i in an interval between T_{start} and T_{end} in EI_j. The failure occurrence probability $F_{P_i}^{\text{EI}_j}$ is calculated by

$$F_{P_i}^{\text{EI}_j} = \frac{\sum_{\text{ext}_k \in \text{EI}_j, P_i} \text{ext}_k^{\text{after failure}}}{\sum_{\text{ext}_k \in \text{EI}_j, P_i} \left(\text{ext}_k^{\text{after failure}} + \text{ext}_k^{\text{after non-failure}}\right)}, \tag{9}$$

where ext_k is the execution task time to invoke an interval k when a price p_i is in EI_j. The after failure function is calculated when the current spot price is above or equal to the user bid. The after non-failure function is calculated when the current spot price is below the user bid.

Using these two thresholds, our scheme performs checkpointing operations in two cases. First, a checkpointing is performed when there is a rising edge in the actual price variation, and the actual price falls in between the user bid and the price threshold; second, the checkpointing is based on the time threshold, which is computed with the failure probability and the average execution time as in (7). It is performed if the current execution time exceeds the time threshold computed at the same past price as the current one.

Algorithm 1 shows the checkpointing and recovery algorithm used in EIC. The flag represents the occurrence of a task failure, and it is initially set to false. The checkpointing process repeats until all tasks are completed. When the task execution is normal (i.e., the flag is false), the scheduler performs a checkpoint operation to cope with the potential job failure (lines 5–25). The scheduler estimates the execution time before the initial task starts (lines 6–9). The recovery process is performed when the flag is true (lines 11–14). The checkpoints are performed in two cases (lines 15–20). If the rising spot price falls in between the user bid and the price threshold, the scheduler performs a checkpointing operation (lines 15–17). If the execution time exceeds the time threshold, the scheduler also performs a checkpointing operation (lines 18–12). When a task failure occurs, the flag is set to true (lines 22–24). Lines 26–29 show the detailed process of time estimation. Lines 30–33 and 34–37 show the detailed process of checkpointing and recovery, respectively.

```
(1)  // Input: user's requested task and bid
(2)  // Output: total task execution time and total cost
(3)  Boolean F_flag = false // a flag representing occurrence of a task failure
(4)  Boolean EI_flag = true // a EI_flag representing a task start
(5)  while (! Task execution finishes) do
(6)      if (EI_flag) then
(7)          Estimation( );
(8)          EI_flag = false;
(9)      end if
(10)     if (spot prices < User bid) then
(11)         if (F_flag) then
(12)             Recovery( );
(13)             F_flag = false;
(14)         end if
(15)         if (rising edge && Price threshold < spot prices) then
(16)         Checkpoint( );
(17)         end if
(18)         if (Time threshold < execution time in current price) then
(19)         Checkpoint( );
(20)         end if
(21)     end if
(22)     if (failure is occurred) then
(23)         F_flag = true;
(24)     end if
(25) end while
(26) Function Estimation( )
(27)     calculate the points of checkpoint to base a price history;
(28)     set the price and time thresholds;
(29) end Function
(30) Function Checkpoint( )
(31)     task a checkpoint on the spot instance;
(32)     Send the checkpoint to the storage;
(33) end Function
(34) Function Recovery( )
(35)     retrieve the checkpoint information form the storage;
(36)     restart the job execution;
(37) end Function
```

Algorithm 1: Checkpointing and recovery algorithm.

5. Performance Evaluation

Our simulations were conducted using the history data obtained from the Amazon EC2's spot instances [19], which was accumulated during a period from December 15, 2010 to December 22, 2010 as shown in Figure 4. The history data before December 20, 2010 were used to extract the expected execution time and failure probability for the proposed checkpointing scheme. The applicability of EIC was tested using the history data after December 20, 2010, which was also used in HBC and REC.

In the simulations, one type of spot instances, c1.xlarge, was applied to show the effect of the three checkpointing schemes on performance according to the user bid and the task time. Table 1 shows the applied resource type details used in Amazon EC2. The high-CPU instance offers more compute units than other resources (standard and high-memory instances) and is ideal for the compute-intensive applications. Under the simulation environments, we compare the performance of EIC with those of HBC and REC in terms of various metrics according to the user bid and task time.

Table 1: Resource type information.

Instance type	Compute unit	Virtual cores	Memory	Storage
C1.xlarge (High-CPU)	20 EC2	8 cores (2.5 EC2)	7 GB	1690 GB

5.1. User Bid Impact on Performance.

5.1. User Bid Impact on Performance. Before analyzing the performance of EIC, we extracted the simulation specifics from the spot history presented in Figure 4. Table 2 shows the data used for simulation. The simulations were conducted with incrementing the user bid interval from minimum bid to maximum bid.

We also extracted the failure probability with the current bid price according to each spot price in the past (12-15-2010–12-19-2010), as drawn in Figure 4. The probability was

Table 2: Simulation parameters and values.

Parameter	Value
Task time	259,200 sec
Checkpoint time	300 sec
Recovery time	300 sec
Minimum user bid	\$0.310
Maximum user bid	\$0.340
User bid interval	\$0.005

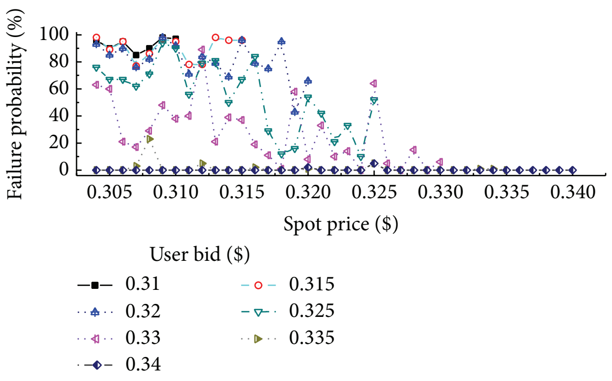
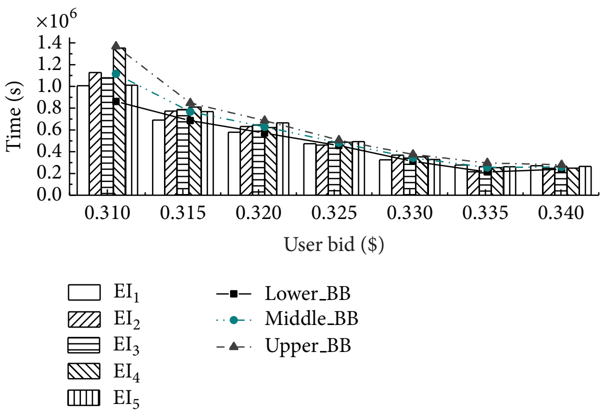
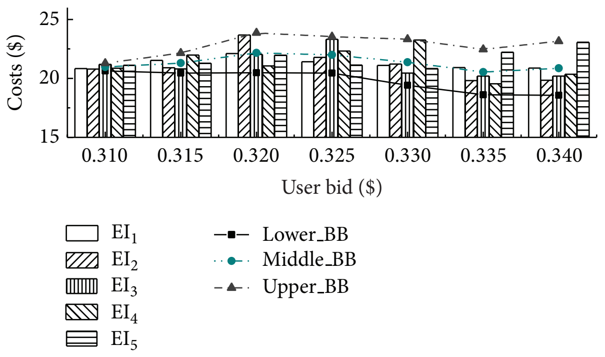


Figure 11: Failure occurrence probability.



(a) Estimated Bollinger Bands of execution time



(b) Estimated Bollinger Bands of costs

Figure 12: Estimated execution time, cost, and Bollinger Bands of each EI zone computed with the past price history.

used to determine the time threshold in EIC. Figure 11 shows the failure occurrence probability for the c1.xlarge instance. The X-axis and Y-axis denote the spot price and the failure probability for a given user bid, respectively.

Figure 11 states that the failure occurrence probability changes according to the user bid. As anticipated, if the bid price is low, the failure probability is high across all spot prices. If the bid price is high, the failure probability is low. Thus, it is reasonable to predict that the task execution time will be longer if the failure probability is high because both the total failure time and total rollback time increase.

Figure 12 estimated execution time, cost, and Bollinger Bands of each EI zone computed with the past price history. Figures 12(a) and 12(b) show the execution time and the cost according to the user bid respectively. Estimated interval (EI) with the weighted moving average which is calculated by using the past spot price history, is necessary for the user bid. Figure 12 also shows the Bollinger Bands (Lower_BB, Middle_BB, and Upper_BB) according to the user bid.

Figure 13 shows the task execution time, the cost, and Bollinger Bands when the number of EI zones is increased. For example, 2 in x-axis means that two zones (EI_1 and EI_2) are included in the simulation.

Figure 14 shows the rollback times of EIC, HBC, and REC. The rollback time is calculated from a failure point in time to the last checkpointed time. The EIC lessens the average rollback time by 72.46% over HBC and 88.49% over REC.

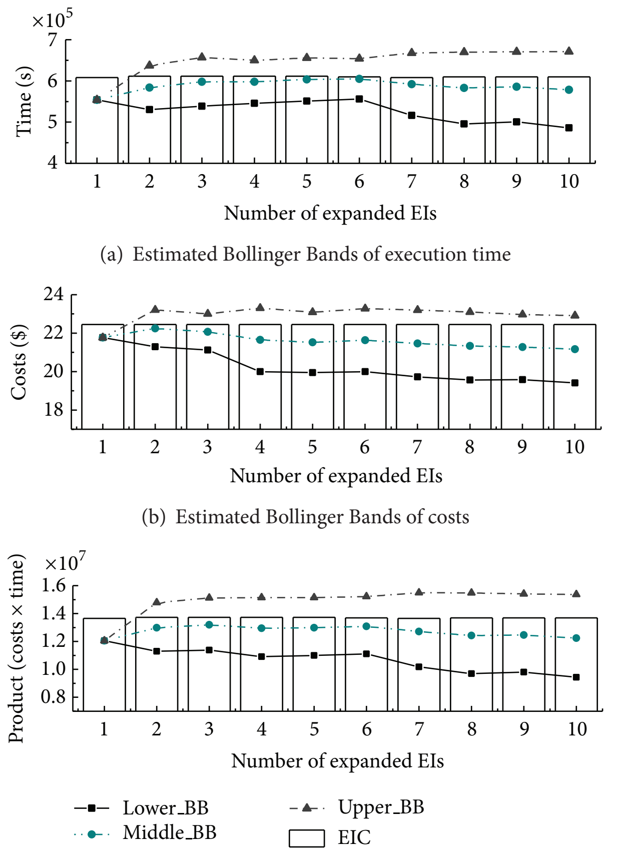
Figure 15 shows the performance comparison of EIC, HBC, and REC. The EIC reduces the number of checkpoints on average by 35.97% and 37.92%, compared to HBC and REC, respectively. Consequently, the EIC shortens the task execution time by 35.53% over the HBC and 40.40% over REC.

Figure 16 shows the total costs according to the user bid. The EIC reduces the cost on average by 36.26% and 38.52% over HBC and REC, respectively.

Figure 17 shows the combined performance metric, the product of the total execution time, and cost. According to the user bid, the EIC shows marginal variation due to the lowest amount of rollback time among the compared schemes. The EIC achieves the relative benefits in the combined metric on average by 55.73% and 60.95% when compared to HBC and REC, respectively.

Figure 18 shows how well the actual execution time and cost are predicted with EIC according to the user bid. The actual execution time and cost are located between the lower and upper bounds of the Bollinger Band. Figures 18(a) and 18(b) show that they are close to the middle point of the Bollinger Band. The experiments show that the adoption of the Bollinger Band would provide reliable estimations to Cloud users.

5.2. Task Time Impact on Performance. In this section, we analyzed the performance of computing-type instances according to the task time. Table 3 shows the simulation parameters. Note that the execution time in simulations



(a) Estimated Bollinger Bands of execution time

(b) Estimated Bollinger Bands of costs

(c) Estimated Bollinger Bands of product

FIGURE 13: Estimated execution time, cost, and Bollinger Bands of expanded EI zones computed with the past price history.

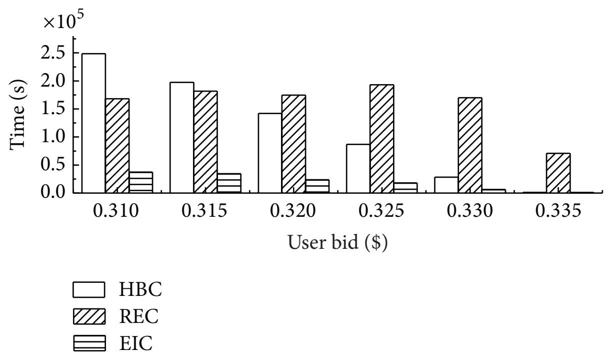
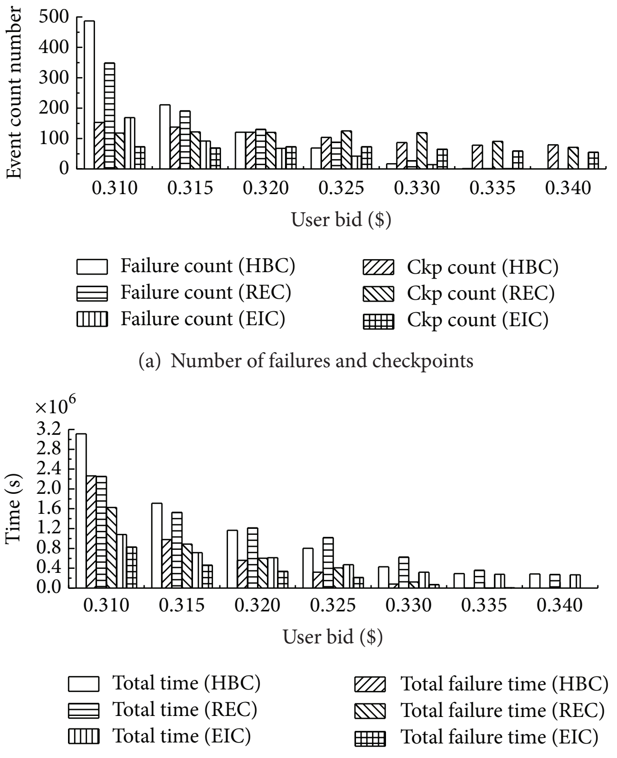


FIGURE 14: Comparison of rollback time according to user bid.



(a) Number of failures and checkpoints

(b) Total task execution time and total failure time

FIGURE 15: Performance comparison of checkpointing schemes according to user bid.

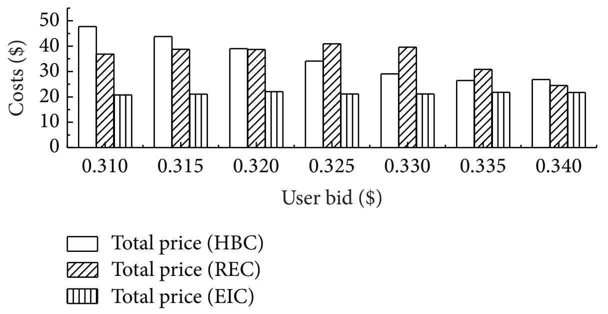


FIGURE 16: Comparison of total costs.

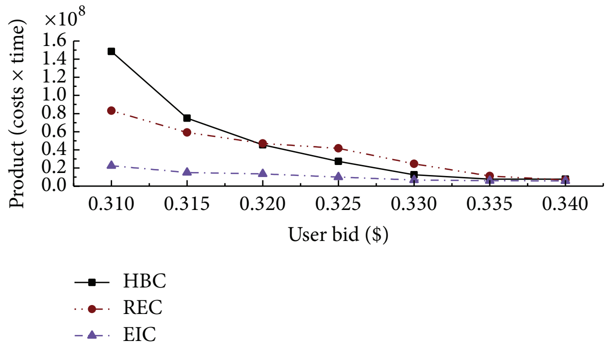


FIGURE 17: Comparison of combined metrics (total task execution time and costs).

varies from minimum time to maximum time at the granularity of the time interval.

Figure 19 shows the rollback time of EIC, HBC, and REC according to the task time. The increase rate of rollback time in EIC is small compared to HBC and REC. The rollback times are increased by 5.25 times, 15.84 times, and 12.41 times for EIC, HBC, and REC, respectively, when the task times are increased from the minimum to the maximum times. EIC

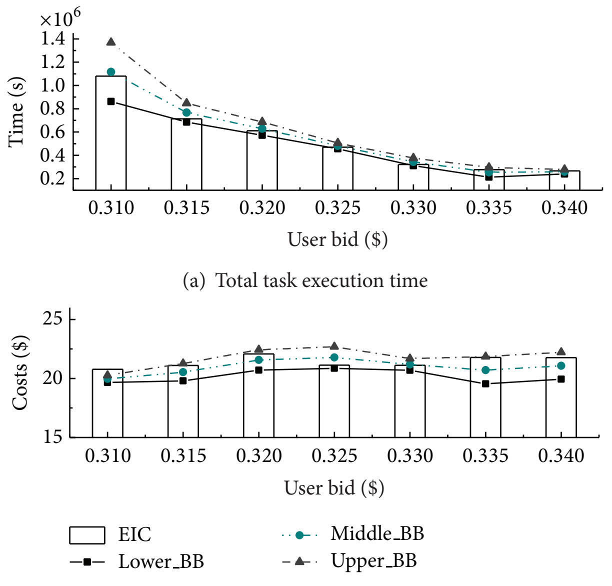


Figure 18: Comparison of actual EIC outputs (execution time and cost) and estimations according to the user bid.

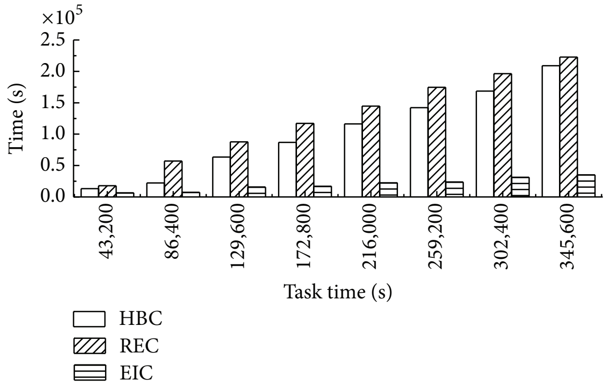


Figure 19: Comparison of rollback times according to the task time.

Table 3: Simulation parameters and values.

Parameter	Value
User bid	\$0.32
Checkpoint time	300 sec
Recovery time	300 sec
Minimum task time	43,200 sec
Maximum task time	345,600 sec
Task time interval	43,200 sec

lessens the rollback time on average by 80.61% and 84.36% over HBC and REC, respectively.

Figure 20 shows the performance comparison of EIC, HBC, and REC. Figures 20(a) and 20(b) show the numbers of failures and checkpoints, and total task execution time and total failure time according to the task time. The EIC reduces the number of checkpoints on average by 31.97% and 32.93% compared to HBC and REC, respectively. Thus, the EIC achieves performance improvements in the task execution time on average by 43.79% and 48.25% over HBC and REC, respectively.

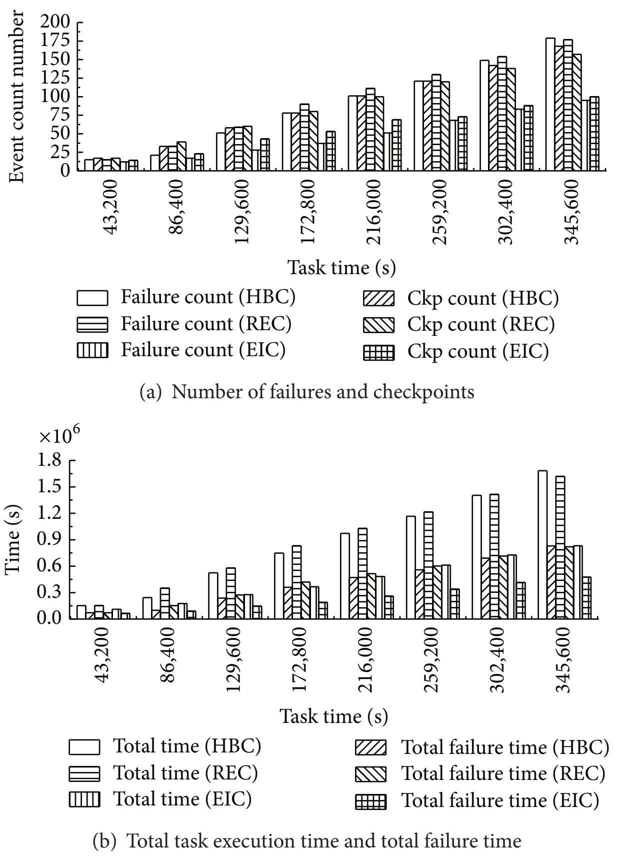


Figure 20: Performance comparison according to the task time.

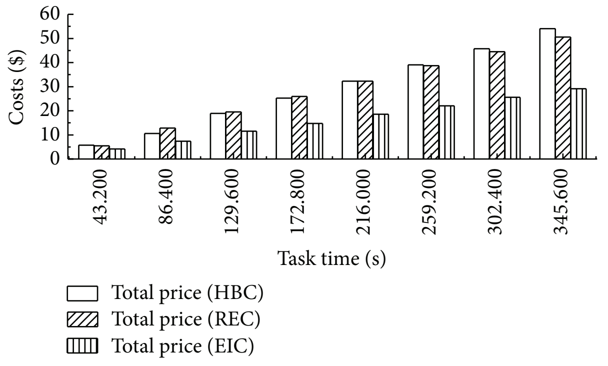


Figure 21: Comparison of total costs.

Figure 21 shows the total cost according to the task time. The EIC reduces the cost on average by 39.38% and 40.08% compared to HBC and REC, respectively.

Figure 22 shows the combined performance metric, the product of the total task execution time, and cost. The rate of increase in the product in EIC is lowest among the compared schemes. The EIC achieves a performance improvement on

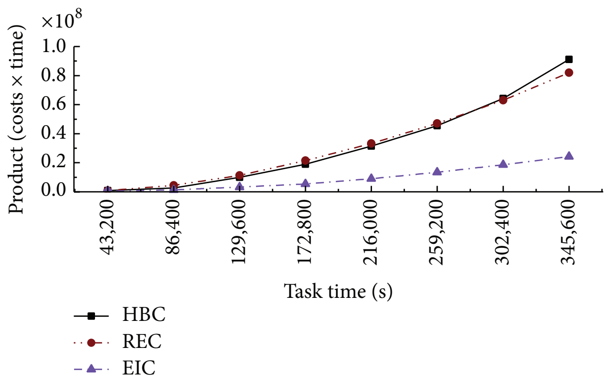


FIGURE 22: Comparison of combined metrics (product of total execution time and cost).

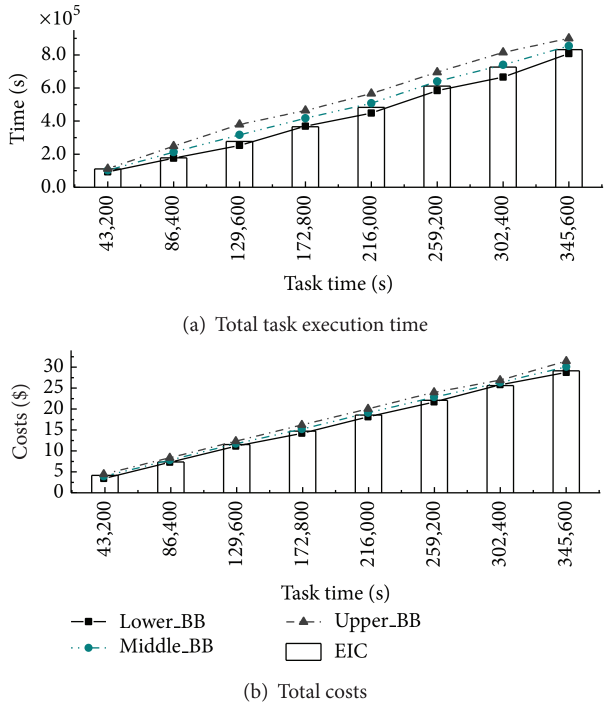


FIGURE 23: Comparison of actual EIC outputs (execution time and cost) and estimations according to the task time.

average by 65.36% and 68.51% when compared to HBC and REC, respectively.

Figure 23 shows the estimation accuracies according to the task time. The actual execution time and cost are located between the lower and upper bounds of the Bollinger Band. Figures 23(a) and 23(b) prove that the actual execution time and cost are close to the middle point of the Bollinger Band. They state that the EIC would be able to offer approximate ranges of total costs and task execution time to Cloud users.

Overall, the EIC significantly reduces the number of checkpoint trials compared to the existing checkpointing schemes. Furthermore, the rollback time is much lesser because the EIC adaptively performs the checkpointing operation according to the execution time and price. Simulation results showed that our scheme achieved the cost-efficiency by reducing rollback time regardless of the resource types of spot instances.

Analyzing history to compute the estimated interval requires overheads such as CPU time. However, computations only involve failure probability, execution time and cost estimations, and a range of the Bollinger Band. Considering the advancement of modern computers, we strongly believe it would take the minimal amount of overheads for computations.

6. Conclusion

In this paper, we proposed the estimated interval-based checkpointing (EIC) in the unreliable cloud computing environment. The weighted moving average estimates the execution time and cost using the price history of spot instances to improve the performance and stability of task processing. The EIC performs the checkpointing operation, based on price and time thresholds. The thresholds are determined based on the moving average and the failure probability. They are used to determine the checkpointing position to recover from the potential failures of spot instances arising from the price fluctuation. The Bollinger Bands determines the lower and upper bounds of the estimated execution time and cost. The ranges are informed to users as guidance for their decision. The simulation results reveal that, compared to the hour-boundary checkpointing (HBC) and rising edge-driven checkpointing (REC), the EIC reduces the number of checkpoints by 35.97% and 37.92%, respectively, on average according to the user bid. It also reduces the rollback time by 72.46% and 88.49% on average. Consequently, the task execution time is decreased with ETC by 35.53% over HBC and 40.40% over REC. The EIC also provides the benefit of the cost reduction by 36.26% over HBC and 38.52% over REC, on average.

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 Korea government (MEST) (NRF-2012R1A2A2A 02046684).

References

[1] 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, September 2008.

[2] 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, November 2008.

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

[4] M. M. Weng, T. K. Shih, and J. C. Hung, “A personal tutoring mechanism based on the cloud environment,” *Journal of Convergence*, vol. 4, pp. 37–44, 2013.

[5] 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.

[6] 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)*, pp. 357–362, October 2009.

[7] Elastic Compute Cloud (EC2), 2014, http://aws.amazon.com/ec2.

[8] GoGrid, 2014, http://www.gogrid.com.

[9] FlexiScale, 2014, http://www.flexiscale.com.

[10] I. Goiri, F. Julià, J. Guitart, and J. Torres, “Checkpoint-based fault-tolerant infrastructure for virtualized service providers,” in *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS ’10)*, pp. 455–462, April 2010.

[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 (NPC ’11)*, pp. 185–200, 2011.

[13] 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.

[14] G. Singer, I. Livenson, M. Dumas, S. N. Srirama, and U. Norbisrath, “Towards a model for cloud computing cost estimation with reserved resources,” in *Proceedings of the 2nd ICST International Conference on Cloud Computing (CloudComp ’10)*, Springer, Barcelona, Spain, October 2010.

[15] M. Mazzucco and M. Dumas, “Reserved or on-demand instances? A revenue maximization model for cloud providers,” in *Proceedings of the 4th IEEE International Conference on Cloud Computing (CLOUD ’11)*, pp. 428–435, July 2011.

[16] W. Voorsluys and R. Buyya, “Reliable provisioning of spot instances for compute-intensive applications,” in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA ’12)*, pp. 542–549, March 2012.

[17] Q. Zhang, E. Gürses, R. Boutaba, and J. Xiao, “Dynamic resource allocation for spot markets in clouds,” in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE ’11)*, pp. 1–6, 2011.

[18] Amazon EC2 spot Instances, 2014, http://aws.amazon.com/ec2/spot-instances.

[19] Cloud Exchange, 2014, http://cloudexchange.org.

[20] A. Andrzejak, D. Kondo, and S. Yi, “Decision model for cloud computing under SLA constraints,” in *Proceedings of the 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS ’10)*, pp. 257–266, August 2010.

[21] P. Patel, A. Ranabahu, and A. Sheth, “Service level agreement in cloud computing,” in *Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications*, pp. 212–217, 2009.

[22] G. Dagnino, “Technical analysis, the markets and moving averages,” Tech. Rep., The Peter Dag Portfolio Strategy & Management, 2013.

[23] R. J. Hyndman, “Moving averages,” Tech. Rep., Department of Econometrics and Business Statistics, Monash University, 2009.

[24] J. Bollinger, *Bollinger on Bollinger Bands*, McGraw Hill, 2002.

[25] Daytrader, “Bollinger bands as an entry technique,” 2000.