Journal of Current Science and Technology, May-August 2023 Vol. 13 No. 2, pp. 412-427

ISSN 2630-0656 (Online)

Cite this article: Tomar, S., Mishra, A. K., & Yadav, D. K. (2023, May). Knowledge-based checkpointing strategy for spot instances in cloud computing. *Journal of Current Science and Technology*, *13*(2), 412-427. https://doi.org/10.59796/jcst.V13N2.2023.1754

Journal of Current Science and Technology

Journal homepage: https://jcst.rsu.ac.th

Knowledge-based checkpointing strategy for spot instances in cloud computing

Sumit Tomar, Ashish Kumar Mishra*, and Dharmendra K Yadav

Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, U. P. India, 211004

*Corresponding Author; E-mail: ashish.rcs51@gmail.com

Received 29 August 2022; Revised 21 February 2023; Accepted 16 March 2023;
Published online 15 July 2023

Abstract

The Amazon EC2 offers spot-priced virtual machines (VMs) at a reduced price compared to on-demand and reserved VMs. However, Amazon EC2 can terminate these VMs anytime due to the spot price and demand fluctuation. Using spot VMs results in a longer execution time and disrupts service availability. Users can use fault-tolerant techniques such as checkpointing, migration, and job duplication to mitigate the unreliability of spot VMs. In this paper, a knowledge-based checkpointing strategy is proposed to minimize the overall checkpointing overhead during the execution of jobs. The proposed scheme uses real-time price history to decide when to take a checkpoint. Results show that the proposed approach can significantly reduce the turnaround time by 18% compared to Hourly Checkpointing Strategy and 9% compared to Rising-Edge Checkpointing Strategy. One can also achieve 54% to 78% reliability with a cost saving of 78% for the workload used with the described approach.

Keywords**:** *checkpointing; cloud computing; fault tolerance; spot instances*

1. Introduction

The evolution of cloud computing has made the long-held dream of utility computing a reality by offering resilient computing resources, platforms, and software as services using the pay-per-use model (Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009). Cloud providers provide raw computing resources, i.e., computing capacity, storage, and network as services in the form of Virtual Machines (VMs), which can also be called Infrastructure-as-a-Service (IaaS). Cloud providers such as Amazon EC2 (Amazon Web Services, Inc., n.d.a), IBM Cloud (IBM, n.d.), and Google Cloud Platform (Google Cloud, n.d.) made VMs available with different subscription models to meet the computing requirements of a broad range of applications. Despite having equivalent computing power, the price varies significantly across available subscription models for the same type of VMs.

Based on subscription models, VMs can be categorized into three types: on-demand, reserved, and spot VMs (Amazon Web Services, Inc., n.d.b). Amazon Web Service (AWS) introduced spot instance (The terms “instance” and “Virtual Machine (VM)” are used interchangeably in this literature) in December 2009 to minimize the operational cost of unused computing capacity. The Google Cloud Platform (Google Cloud, n.d.) has recently presented preemptible VM instances similar to AWS's spot VMs. As mentioned earlier, on-demand instances are provisioned instantly and have the highest price among the three types. Reserved instances are offered at reduced prices but

with a long-term commitment. In the case of spot instances (SIs), cloud providers permitted users to bid for the resources and made spot VMs available to them until the bid price remained higher than the spot price. Spot prices vary dynamically based on demand and supply at a data center.

Figure 1 depicts the variation in spot price within “us-west-1” region for “c1.xlarge" type of instance. If the spot price exceeds the user bid, the cloud provider can reclaim the SI without warning. Users can significantly reduce monetary costs by choosing spot VMs over on-demand and reserved instances to compute intensive divisible workloads such as video encoding, testing, data processing, web crawling, and scientific research (Amazon Web Services, Inc., n.d.b). Spot instances are available at a reduced price, but reliability risk is associated with them due to abrupt termination over irregular intervals. The intermittent nature of SIs results in disruption of task execution and increased turnaround time.

Various research studies have proposed fault-tolerant techniques, including migration, checkpointing, and job duplication, to tackle the inherent unreliability of computing systems (Yi, Andrzejak, & Kondo, 2012; Yi, Kondo, & Andrzejak, 2010; Wang, Huang, Vo, Chung, & Kintala, 1995; Voorsluys, & Buyya, 2012; Jangjaimon, & Tzeng, 2015; Goiri, Julia, Guitart, & Torres, 2010; Hussain, Znati, & Melhem, 2019; Yang, Khuller, Choudhary, Mitra, & Mahadik, 2021). Checkpointing helps reduce task turnaround time and cost by saving the state of a process in execution periodically to reliable storage from where it can be restored in the future (Wang et al., 1995). For SIs, the process can be restarted from its last checkpoint or the saved state when the instances become available after an out-of-bid situation. Checkpointing can be done either at the system level or the application level. Both approaches have their advantages and disadvantages.

- **Application level:** In this technique, checkpointing and recovery operation is inserted in the application code so that the application stores and recovers its state whenever required. It does not hold the entire state of the system but the state of application only. Application-initiated checkpointing is more portable and efficient than system-level checkpointing. However, due to the setting in the application code, checkpointing operations can be performed only at fixed time intervals.
- **System level:** Checkpointing is carried out at the level of the system, which is used to execute the application. It stores the state of the application along with the system state. It consumes more storage space as compared to the application level checkpointing. It facilitates users to take checkpoints at arbitrary intervals during the application's lifetime. It can be effective when kernel-level information is required during the recovery process.

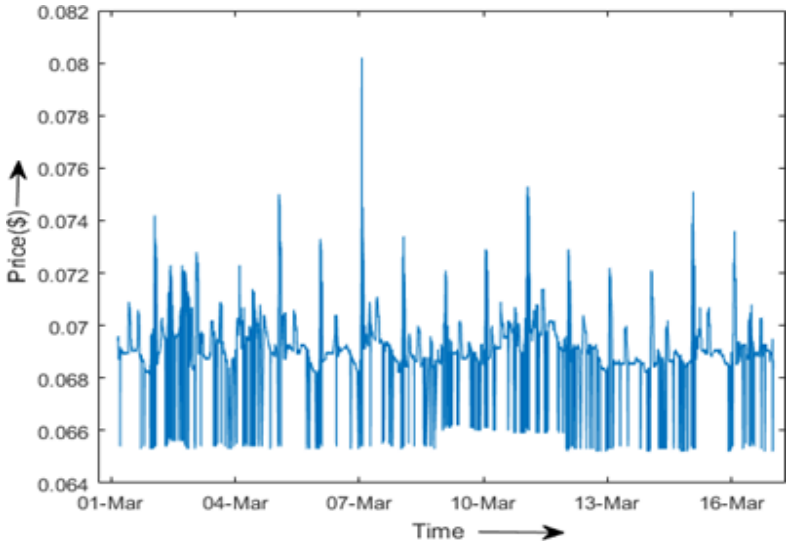

Figure 1 Price variation for us-west-1, c1.xlarge instance type

Users can utilize spot instances with in-memory checkpoints to increase reliability and reduce the time and cost of job execution (Hussain et al., 2019). These instances are further used for reducing the cost of machine learning task training (Yang et al., 2021), along with the utilization of efficient checkpointing algorithms. If the in-memory checkpoint node fails, recovery is not possible. Hence, users can recover from the checkpoint stored on the disk. Therefore, there is a need to design an efficient checkpointing algorithm that reduces the overhead of taking checkpoints and reduces the loss due to the revocation of spot instances.

The rest of the paper is structured as follows. Section 1.1 highlights the literature review on resource provisioning and utilization in spot markets. Section 2 highlights the main objective of the research. The methodology or System model used to run applications with fault-tolerant mechanisms is described in Section 3. The description of the devised checkpointing strategy is presented in Section 4. The correctness of the proposed algorithm is shown in Section 5. The Result and Discussion section compares the design process's performance with other schemes (Section 6). Finally, Section 7 presents the conclusion of the paper.

1.1 Related Work

Several attempts have been made to optimize the cost of scheduling applications on cloud resources (Chohan et al., 2010; Mattess, Vecchiola, & Buyya, 2010; Popovici, & Wilkes, 2005; Song, Zafer, & Lee, 2012; Wu, Garg, & Buyya, 2012). Users can minimize job turnaround time and monetary charges using an optimal bidding strategy or an efficient fault-tolerant mechanism.

Authors utilize the SIs to reduce the runtime of MapReduce tasks (Chohan et al., 2010). They have predicted the availability of SIs using the Markov chain. They describe using a fault-tolerant technique to overcome the adverse effect of SIs termination on running the MapReduce jobs.

Hybrid cloud architectures can offload the peak load of an in-house cluster on SIs in the public cloud (Mattess, Vecchiola, & Buyya, 2010). They also examined the trade-off between cost conservation and several deadlines violations while using SIs compared to on-demand VM instances. The problem of job scheduling on the resources offered with uncertainty regarding their price variations and availability over time is addressed (Popovici, & Wilkes, 2005).

Various attempts in literature have been made to analyze the behavior of Amazon's spot pricing policy (Amazon Web Services, Inc., n.d.b) to develop price prediction models and bidding strategies (Agmon Ben-Yehuda, Ben-Yehuda, Schuster, & Tsafrir, 2013; Andrzejak, Kondo, & Yi, 2010; Javadi, Thulasiramy, & Buyya, 2011). The authors used the spot price history provided by Amazon. Different variations of the Markov Chain Model are used to examine the variations in spot price (Chohan et al., 2010; Song, Zafer, & Lee, 2012; Jangjaimon, & Tzeng, 2015). In the article (Agmon Ben-Yehuda, Ben-Yehuda, Schuster, & Tsafrir, 2013), the authors disassemble the approach of spot pricing utilized by Amazon to examine the same. They have claimed that current demand does not affect the spot price but is generated randomly by an internal reserve price mechanism.

There has been constant work on using different variants of fault-tolerant techniques to reduce the monetary cost and turnaround time while using Sis (Goiri, Julia, Guitart, & Torres, 2010; Jangjaimon, & Tzeng, 2015; Yi, Kondo, & Andrzejak, 2010; Yi, Andrzejak, & Kondo, 2012; Voorsluys, & Buyya, 2012). The authors proposed an intelligent checkpointing-based infrastructure to reduce the checkpointing overhead (Goiri, Julia, Guitart, & Torres, 2010). They used Union File System to speed up the checkpointing process and Hadoop Distributed File System to store checkpoints for efficient recovery. They have demonstrated the effectiveness of the checkpointing mechanism in reducing job execution time.

An adaptive checkpointing scheme to reduce the number of checkpoints against the primary checkpoint method (hour-boundary and rising edge-driven) is proposed by the authors in the paper (Yi, Kondo, & Andrzejak, 2010). After a constant time, interval, the proposed strategy decides whether to take or skip a checkpoint based on estimated recovery time. In the article (Yi, Andrzejak, & Kondo, 2012), the authors proposed a modified version of the existing checkpointing schemes. The decision to take or skip a checkpoint depends on the current spot price and failure probability of associated VMs.

A formula is derived for calculating a job's required number of checkpoints, and an algorithm is proposed to minimize the execution cost concerning

checkpointing overhead (Di, Robert, Vivien, Kondo, Wang, & Cappello, 2013). The algorithm is dynamic enough to be adaptable, with a variable remaining workload and a varying failure probability. Evaluation of the approach is done in real environments with hundreds of VMs.

The concept of Elastic Spot Instances has been suggested in which customers interrupt the instances (Dawoud, Takouna, & Meinel, 2012). In this strategy, when the spot price is more in comparison to the user's bid price, instead of abruptly terminating the spot instances, the provider scales down the capacity allocated to the instances in proportion to the price increase. By eliminating the fragmentation of allocation (only full billable hours are now allocated instead of interrupting an execution in a middle of an hour time frame), this approach increases the provider's revenue. The said approach encourages the checkpointing strategy to the optimum level.

The authors provide an adaptive incremental checkpointing (AIC) strategy, which reduces the size of checkpoint files so that the overhead of checkpointing is lowered, decreasing the turnaround time (Jangjaimon, & Tzeng, 2013). The paper gives the concept of multilevel checkpointing with delta compression. They develop a Markov model to predict the performance of multilevel concurrent checkpointing. AIC utilizes an idle core from multicore systems for concurrent checkpointing.

A checkpointing scheme based on price history has been suggested to reduce the task completion period (Jung, Chin, Chung, Yu, & Gil, 2011). The proposed method depends on SLA to satisfy the user-level requirements. The authors calculated the price band to decide the suitable time for the checkpoint. However, their price indicator calculation differs from KBCS (Knowledge-Based Checkpointing Strategy). They have not considered the mean time between failure in the proposed technique.

Authors propose an online extended consensus revenue estimation scheme in a recurrent, multi-unit, and single-price auction for IaaS cloud resources (Toosi, Vanmechelen, Khodadadi, & Buyya, 2016). The said approach is envy-free. The suggested approach is combined with the mechanism for computing reserve prices dynamically based on the power usage effectiveness of data centers and electricity costs. It is shown how the said approach improves the classical auction by simulation-based evaluation. To maximize profit, the authors value the history of VM's execution time. The proposed mechanism can get optimal revenue without requiring the history of bid distributions. The authors also provide an experimental study with a system prototype that confirms the validity of the proposed approach in the real world.

An algorithm is designed for the execution of Hadoop systems in a dynamic public cloud (Chen, Lee, & Tang, 2014). This algorithm takes advantage of spot instances to improve the efficiency of active Hadoop systems. Authors propose auto scaling of VMs with migration algorithm to avail spot instances in the cloud. An experimental evaluation of the proposed strategy has been provided to prove that the algorithm can improve efficiency by a factor of 9.3x.

Authors analyze SIs based on one-year price history at four data centers of Amazon's EC2 (Javadi, Thulasiram, & Buyya, 2013). They analyze the different SIs in respect of spot prices and the time between price changes to ascertain the time dynamics for the spot price in terms of hour-in-day and day-of-week. These two data series have been validated using a statistical model. The model is proposed with Gaussian distribution using three or four components of eight types of SIs. Validation of the model through simulation is done to prove that the proposed model accurately predicts the total cost of an active job on SIs.

In the article, "Checkpointing as a service: enabling application-level checkpointing and migration in diverse cloud environments" (Cao, Simonin, Cooperman, & Morin, 2015), the authors propose a novel approach to facilitate application-level checkpointing and migration in various cloud environments. The approach depends on a mechanism for adding fault tolerance to the present cloud platform. The external checkpointing package is not dependent on the object platform, which is being used to get cloud-agnostic properties by the pro- posed approach. The devised cloud-agnostic checkpointing service (CACS) is validated through two cloud platforms: Snooze and OpenStack. The CACS is designed to provide a single checkpoint service for different cloud platforms, and it also supports migration from classical environments to the cloud.

In the paper titled "Dynamic Resource Allocation Strategy for Spot Instances in Cloud Computing" (Sharma, Irwin, & Shenoy, 2016), authors suggest that when the spot instance price

becomes greater than the bidding price, users should search for resources elsewhere to execute the jobs despite waiting. The author's focus is not on optimizing bidding strategies but on modifying applications to search and move to low-cost resources. Thus, the authors tried to break the notion that the bidding strategies mainly affect the availability and cost of SIs. Spot price history shows that the availability and cost of SIs are mainly constant across a broad range of bids.

In the article of Mishra, Umaro, & Yadav, (2018), a detailed literature survey has been provided. In the paper, it is suggested that fault tolerance of task execution can be improved by checkpointing the tasks' progress at the optimal time, and for increasing the reliability of task execution on these instances, the bidding time of spot instances should be carefully chosen. A decision-based checkpointing strategy (DBCS) has been proposed in the paper (Mishra, Yadav, Kumar, & Jain, 2019b), using the machine learning technique for spot price prediction. The DBCS strategy is suitable only for divisible workloads into independent modules.

A price prediction technique using kNN regression has been proposed by Liu, Wang, Meng, Zhao, & Zhang, (2020) for predicting the price of spot instances and bidding accordingly. The authors compared the performance of their proposed method with some other machine learning techniques. With the price prediction in advance, one can bid accordingly to retain the instance for the maximum time. Whenever the predicted price exceeds the user's limit, the task progress can be checkpointed using our proposed KBCS.

According to Ramesh, Pradhan, & Lamkuche, (2021), the utilization of different cloud resources, their pricing mechanisms, and models. Fifteen pricing models have been analyzed to benefit consumers and providers. Authors have claimed that many more models need to come at the implementation level to provide profits to consumers and providers. A suitable pricing mechanism can be combined with the proposed checkpointing scheme to increase users' profit.

A detailed survey on the use of preemptible cloud resources has been provided in the article of Deldari, & Salehan, (2021). Authors have presented several issues and challenges in using such types of resources. In the survey, it was claimed that the previous checkpointing schemes did not consider the utilization of spot instances. In contrast, our proposed scheme is solely based on the utilization of spot instances.

The paper has predicted the price of spot instances using neural network techniques (Agarwal, Mishra, & Yadav, 2017). The technique uses the concept of recurrent neural networks for prediction. The limitation of the technique is that it is not adaptable to minor changes. One can combine the proposed checkpointing algorithm with the price prediction algorithm to have combined and better results.

Mishra, Kesarwani, & Yadav, (2019a), proposed an approach for predicting the short-term price of spot instances has been devised. The price is predicted using the time series and probabilistic mechanism. The past prices of the instances have been used in the prediction. The algorithm is flexible enough to incorporate variations in the granularity of seconds or minutes. The proposed checkpointing algorithm can be combined with the price prediction algorithm and checkpoint the task whenever the predicted price lies above the predefined range.

According to Hussain, Znati, & Melhem, (2019), an approach for in-memory checkpoints was proposed. Time to the checkpoint can be reduced by taking checkpoint in-memory, not in the disk. However, in case of failure of in-memory checkpointed nodes, which may occur for any reason, recovery is impossible. In such a situation, one has to recover from the checkpoint, which is stored on a disk.

Alourani, & Kshemkalyani, (2020) proposed a mechanism for utilizing spot instances without any fault tolerance techniques in the article. The authors claim that the mechanism reduced the cost and time compared to the utilization of these instances with fault tolerance mechanisms. It has been suggested that with the employment of fault tolerance mechanisms, the overhead increased, increasing cost and time. The proposed checkpointing technique will be helpful when the loss due to the revocation of instances is more compared to the overhead that occurred in the involvement of the proposed checkpointing algorithm.

According to Yang, Khuller, Choudhary, Mitra, & Mahadik, (2021), the spot instances are scheduled for training machine-learning jobs to reduce the cost of training. The authors have devised a linear programming-based polynomial time algorithm for analyzing the trade-off of computations that are interruptible/low-cost and

uninterruptible/high-cost. The authors claim that with the technique, they can reduce the budget from 23% to 48% compared to the instances of the on-demand type. They also support the utilization of checkpointing techniques to get better results in the use of spot instances.

We can observe from the state-of-the-art survey that only a few authors have used real price history to propose a checkpointing scheme. Nevertheless, they have not considered MTBF (mean time between failure) to devise their schemes. All the other checkpointing techniques combine two fundamental techniques, namely Hourly Checkpointing and Rising-Edge Checkpointing strategies. Hence, we have compared our work only with these two basic schemes.

2. Objectives

The paper's main contribution is to propose a checkpointing strategy to decide the suitable time to checkpoint the tasks' progress. More checkpoints will avoid loss due to revocation events, but at the same time, it will increase the overhead of checkpointing. The number of checkpoints may increase the loss during the revocation event. Hence, a balance in the number of checkpoints is required.

In the proposed approach, checkpointing refers to system-level checkpointing since the checkpointing operations are not performed at fixed time intervals. Even in the case of intermittent failure due to out-of-bid situations, it is profitable to employ SIs to run compute-intensive jobs at a minimal price compared to on-demand and reserved VMs. This article uses a knowledge-based checkpointing scheme to minimize the number of checkpoints to reduce overall checkpointing overhead during task execution. The spot price history provided by Amazon EC2 (Amazon Web Services, Inc., n.d.a) is used to decide when to take a checkpoint during the lifetime of a task. Simulation results in Section 6 show that the proposed strategy outperforms the fundamental checkpointing strategy, namely the hourly and rising-edge checkpointing strategies (Yi, Kondo, & Andrzejak, 2010; Yi, Andrzejak, & Kondo, 2012).

3. Methodology (Execution Model)

In this section, the execution model of using spot instances with checkpointing schemes is described. Figure 2 shows how spot instances work with checkpointing. Users submit the spot request with its characteristics (i.e., number of instances, bid price, request type, Etc.). After submission of the request for spot instances, it becomes open to be fulfilled by any VM in the chosen availability zone. If the spot instance price is less than the user's bid price, the request will be fulfilled immediately; otherwise, it will remain open until the spot instance price becomes less than the user's bid price. Once a request becomes active, tasks can be assigned to the instance from the Amazon EBS (Amazon Web Services, n.d.c).

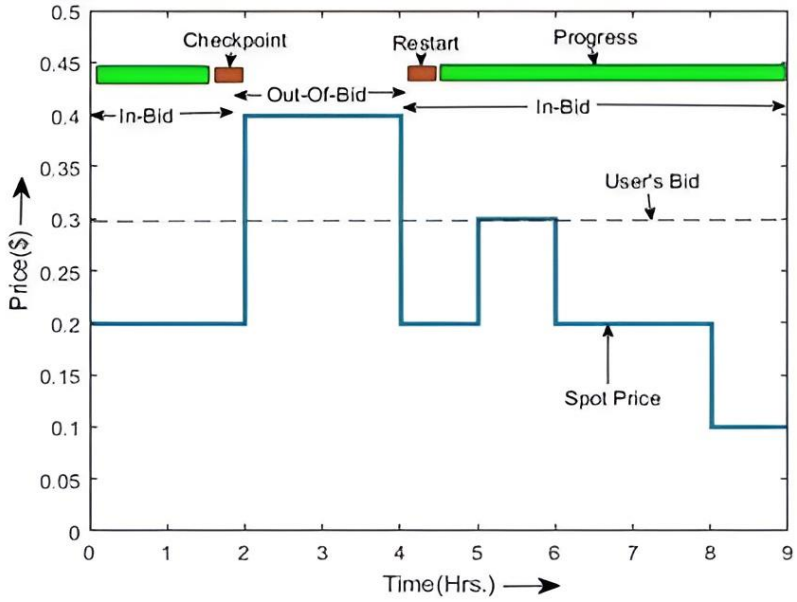

Figure 2 Execution model of spot Instances with checkpointing

Amazon S3 and EBS provide elastic storage for durable and frequently accessed data. Amazon S3 can be accessed across regions, while Amazon EBS is accessible within a region. The storage service selection depends on the diversity of instances and the need for an application. Spot instance is allocated to the user as long as the user's bid price is above the price of the spot instance.

When the spot instance's price surges over the user's bid price, the spot instance is revoked by the cloud provider, which is called an 'out-of-bid' situation. Amazon's spot pricing policy does not charge for partial hours unless the user initiates the termination.

Checkpointing is widely used as a fault-tolerant mechanism with spot instances. It consists of store and recovery operations. The computation progress is stored after an interval, which is decided by the acclimated checkpointing scheme. When the spot instance price goes down by the user's bid price, the task's state is restored from the last checkpoint. However, the checkpointing operations also bring an overhead because the progress of tasks is paused during the checkpointing process. The total cost of the operation can be computed by considering the checkpointing and recovery overhead in case of failure. It also must consider the number of checkpoints and the number of times the recovery operation is performed. So the total cost of the operation can be defined as:

$$\text{Total Cost} = \sum_{i=t}^{T} P_i + c * T_c + r * T_r \qquad (1)$$

Where T is the total time (in hours) for a job to execute on a selected spot instance, Pi is the spot price (in $) during the i^{th} hour of task execution. Tc and Tr are checkpointing and recovery overhead, respectively. The unit of Tc and Tr is hours. c is the number of checkpoints, and r is the number of times recovery operation is performed during the lifetime of a job. The checkpointing overhead occurs when the decision is to checkpoint the tasks' progress. If the decision is not to checkpoint the tasks' progress, users must recover the task from the last checkpointed position—the recovery results in recovery overhead. A job can be in various states during its execution. Figure 3 describes the transition of a job through multiple states, i.e., Created (1), Ready (2), Waiting (3), Running (4), Failed (5), and Completed (6).

Once the job is submitted to the broker, which forwards it to the resource provider, it will enter into the ready state. If the VM is available, the broker submits the job to the VM, so now the job will be running. Otherwise, it will be added to the waiting queue. From the waiting state, a job can enter into the running state as soon as the VM becomes available. During the job's execution, the checkpoints will be taken according to the checkpointing strategy. If the job fails due to an out-of-bid situation, the job is moved to the failed state. The broker added the job to the ready queue from the failed state.If the job successfully completes its execution, it will be moved to the completed state.

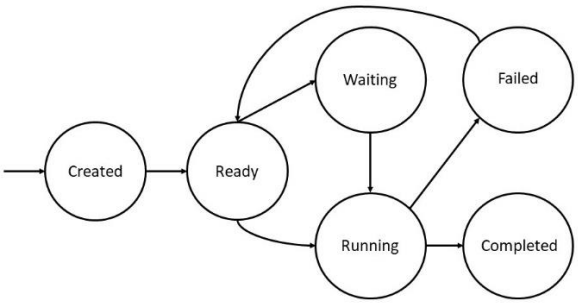

Figure 3 Life cycle of a job

4. Checkpointing Schemes with Spot Instances

In this section, first, the existing checkpointing strategies used for comparison are described briefly. Then, the proposed knowledge-based checkpointing mechanism is presented in detail. It is also described how the proposed approach uses the price history of SIs and weighted moving average to decide the time for checkpointing.

a. Existing Checkpointing Schemes

Various existing checkpointing schemes can be broadly classified into the following four categories:

i. Hourly Checkpointing Strategy (HCS)

In this strategy, the state of the running task is saved after a fixed time interval of one hour from the starting point of execution. Since spot instances are chargedhourly and the partial hour is not to be paid, it's the most instinctive way to create checkpoints at the hour boundary. The recovery time is minimized; however, the number of checkpoints increases for large-size jobs. The large-size jobs are the jobs having longer execution times. So, it requires many hours to complete the job. At each hour, one checkpoint will result in more checkpoints.

ii. Rising-Edge Checkpointing Strategy (RECS)

In this scheme, the checkpoint is taken whenever the spot price increases during the lifetime of the job. An increase in the spot price shrinks the gap betweenthe user bid and the current spot price, consequently increasing the chances of an out-of-bid situation soon. In the case of frequent spikes (i.e., rising edge) in the spot price, RECS results in more checkpoints, while the recovery overhead increases if the spikes occur infrequently.

iii. Checkpointing Combinations

The above checkpointing schemes can be combined to generate other different types of checkpointing strategies. One example is the adaptive checkpointing strategy. This strategy decides whether to take or skip a checkpoint at every hour boundary (for HCS) or at every rising edge (for RECS). This decision seriously impacts the recovery time and hence execution time of a running task in case of failures. Some other examples are adaptive HCS, adaptive RECS, hour boundary, rising edge-driven checkpointing, etc.

b. Devised Checkpointing Scheme

Our designed checkpointing scheme in this article is named a Knowledge-Based Checkpointing Scheme (KBCS). In KBCS, the decision to take a checkpoint is made by analyzing the spot price history of spot instances. The proposed strategy improves the shortcomings of RECS and HCS by considering the rising edges in the spot price, which are above a critical point, estimated using knowledge of price history. The checkpoint_indicator denotes the critical point.

Algorithm 1 shows the steps involved in analyzing the price history and taking the checkpoint accordingly. The weighted moving average (wma) is used to predict the range of spot prices in the upcoming duration. The weighted average is preferred to emphasize the recent spot price variation, which estimates a more accurate bound to decide whether to take or skip a checkpoint. wma_{middle_band} with the user's bid is used to calculate the critical point. Mean-time-between-failure (MTBF) is also incorporated to consider checkpoints near the average lifetime of spot VMs. The MTBF is obtained by dividing the number of operational hours by the number of failures.

Table 1 Notation Table

Symbol	Description
T^{begin}	Beginning time of the learning window
T^{end}	Ending time of learning window
N	Number of Duration in which learning window is divided
D	Length of each duration
T^{begin}	Beginning time of each duration
T^{end}	Ending time of each duration
checkpoint_indicator	Critical point
Wma	Weighted moving average
MTBF	Mean time between failure
min_{Dj}	Minimum price in duration D_i
A	Weight factor used to give more impact on the most recent price

Algorithm 1 Knowledge-Based Checkpointing Strategy

1. ***while*** ($J_{remaining\_time} > 0$ & $t \epsilon [t_{N+1}^{begin}, t_{N+1}^{end}]$)
2. ***if***(*checkpoint_indicator*$_{N+1}$ ***is not*** *set*)
3. ***for*** $\forall D_i, 1 \leq i \leq N$
4. $middle\_band_i = \frac{user\_bid - min_{D_i}}{2}$
5. ***end for***
6. $wma_{middle\_band}^{<T^{begin}, T^{end}>} = \frac{\sum_{i=1}^{N} \alpha_i \, middle\_band_i}{\sum_{i=1}^{N} i}$, where α is the weight factor
7. $checkpoint\_indicator_{N+1} = wma_{middle\_band}^{<T^{start}, T^{end}>}$
8. ***end if***
9. ***if*** ($spot\_price_{t-1} < spot\_price_t$ & $(user\_bid - spot\_price_t) < checkpoint\_indicator_{N+1}$) | $(t\text{-}last\_checkpoint) \geq MTBF$)
10. *take_checkpoint*
11. ***else***
12. *skip_checkpoint*
13. ***end if***
14. ***end while***

Various notations used in Algorithm 1 have been listed in Table 1. In step 1, loop condition,

Jremaining_time > 0
& tϵ[t_(N+1)^begin, t_(N+1)^end],

denotes that the loop will be executed till processing time remains, and t lies in between or at the beginning and end time of each duration. The checking of whether the checkpoint_indicator is being set for the current duration or not is done in Step 2. If it is not set, steps 3, 5, 6, and 7 are executed. A loop will be executed for every duration to calculate the middle band (middle_band), which is computed using the formula mentioned in step 4. The calculation of the weighted moving average (wmamiddle_band) is done in Step 6. In step 7, checkpoint_indicatori is calculated using user_bid and wmamiddle_band. The checking of whether the rising edge is above the critical point or not is done in Step 9; if it is, then the checkpoint is taken; otherwise, the checkpoint is skipped. In this step, it has also been checked that the last checkpoint time is greater than or equal to MTBF; if it is, then the checkpoint is taken; otherwise, the checkpoint is skipped.

5. Correctness of Algorithm

To prove the correctness of Algorithm 1, the following precondition, loop invariant, and postcondition are identified.

Pre-condition (P):
Jremaining_time = Jprocessing_time
and $\boldsymbol{t = t_{begin}^{N+1}}$

Loop-invariant (*I*):
Jremaining_time >0 and checkpoint_indicator (critical point) is set and necessary checkpoints are taken $\forall t \epsilon [t_{N+1}^{begin}, t_{N+1}^{end}]$

Loop-condition (*C*):
Jremaining_time > 0 & $\boldsymbol{t \epsilon [t_{N+1}^{begin}, t_{N+1}^{end}]}$

Post-condition (*Q*):
$J_{remaining\_time}=0 \,||\, t \notin [t_{N+1}^{begin}, t_{N+1}^{end}]$

- The loop invariant *I* is accurate at the beginning, which means that $P \Rightarrow I$, as $J_{remaining\_time} = J_{processing\_time} > 0$, and

since $t = t_{begin}^{N+1}$, so checkpoint_indicator (critical point) is set, and necessary checkpoints are taken $\forall \boldsymbol{t \epsilon [t_{N+1}^{begin}, t_{N+1}^{end}]}$

- *I* and *C* are true before executing *Algorithm*1, and *I* remain true after the I and C are true before executing Algorithm1. I remain true after the execution of Algorithm1 ($I \wedge C \xrightarrow{Algorithm\ 1} I$), as there is no decrement in processing time before the execution of Algorithm1 and initially $\boldsymbol{t = t_{begin}^{N+1}}$.
- When C becomes false, I inferred the postcondition Q ($I \wedge \bar{c} \implies Q$) as $\bar{c}$ means Jremaining_time $\not> 0$ or $t \notin [t_{N+1}^{begin}, t_{N+1}^{end}]$.
- After each loop iteration, the t will get closer to t_{N+1}^{end}, so the remaining number of required iterations decreases.
- If *C* is true, then at least one iteration of the loop remains to be executed, as the postcondition will be valid when C becomes false.

From the above five points, it can be concluded that Algorithm 1 is formally correct.

6. Results and Discussion (Performance Evaluation)

The execution time of the KBCS algorithm is in the order of *N*. There are two loops in the KBCS algorithm. The outer loop will be executed only for the job's remaining time. The inner loop will be executed as many times as the numberof durations in which the learning window is divided. Within the learning window, the complexity will be linear in terms of the number of durations (*N*). For the whole task, the complexity will be *N* *length of the learning window. Here we assume that the task will only be processed within the learning window.

The CloudSim (Calheiros, Ranjan, Beloglazov, De Rose, & Buyya, 2011) simulator has been extended to model the spot instances.

The addition of **SpotBroker**, **SpotDatacenter**, **SpotCharacteristics,** and **VmSpot** entities are performed to the existing CloudSim package.

SpotBroker: The spotBroker component is responsible for job failures, jobadmission, and job execution.

SpotDatacenter: The spotDatacenter component is extended to perform price monitoring, bill generation, and checkpointing operations.

SpotRequestCharacteristics: This entity models the spot request characteristics (i.e., number of instances, bid value, instancetype, type of request, etc.).

VmSpot: The VM entity of the Cloudsim package is extended to support spot VM characteristics (i.e., state, bid price, etc.).

Figure 4 depicts the sequence of events that occurred during the simulation process. After the creation of DatacenterSpot, SpotBroker, Spot Requests, and Cloudlets (entity models the job or task in clouds), a Spot request is submitted to SpotBroker. It requests the spot instances from the DatacenterSpot. When the spot price goes down, the user's bid price and the SIs are assigned to the user. Once the spot instance is assigned, SpotBroker submits the job to the SpotDatacenter.

Various internal events (i.e., price monitoring, bill generation, checkpointing) are processed within the SpotDatacenter during the execution of the job until the out-of-bid event occurs. The time interval between the two price monitoring events is called inter-price time. The checkpointing event is invoked by the checkpointing strategy, which is used within the DatacenterSpot. Since Amazon EC2 charges spot instances on an hourly basis, the cost of using SIs is sent to SpotBroker at a regular interval of one hour. When SI fails due to an out-of-bid situation, the partially completed job is returned to the SpotBroker. This process is repeated until all the jobs are completed. Once all the jobs are completed, they will be returned to the user along with the total cost

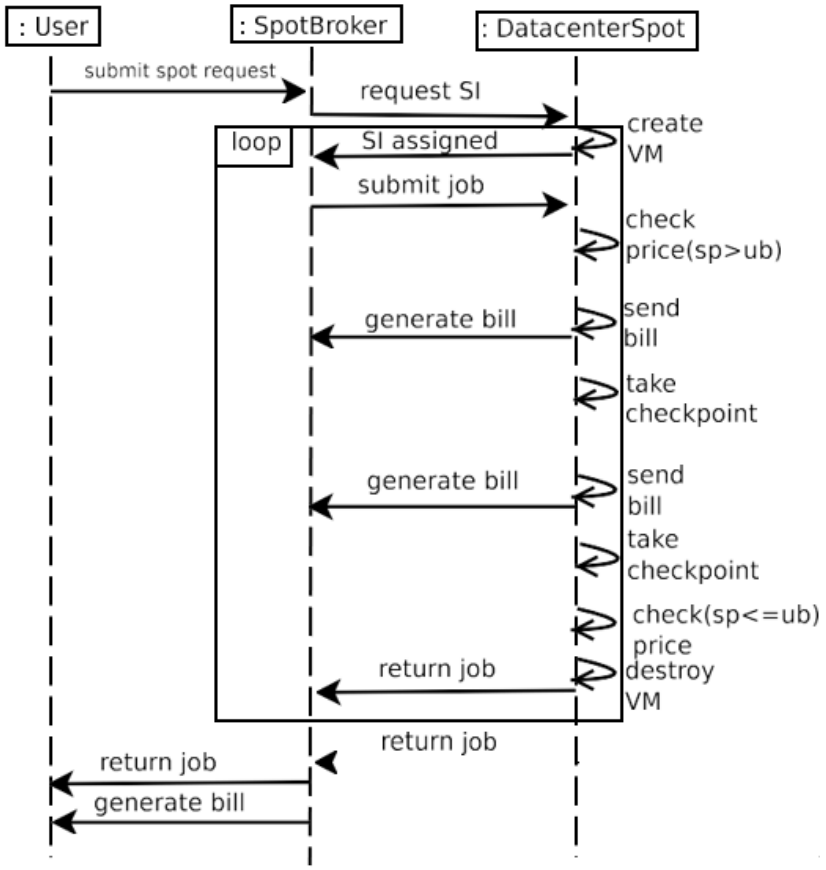

Figure 4 Simulation model

To evaluate the performance of KBCS, the spot price history (Amazon EC2 provides spot price history for the last three months across all data centers. We have used the price history from Dec 2018 to Mar 2019 within the us-west datacenter) is obtained from Amazon EC2 (Amazon Web Services, Inc., n.d.a). The spot price history from Dec 2018 to Mar 2019 is used for the estimation of *checkpoint_indicator* for the upcoming duration. Table 2 shows the minimum price, maximum price, and mean price inspected, during the observation period (i.e., from Dec 2018 to Mar 2019). A near minimum value (0.0152$) is used as bid value ($u_b$) during the simulation. The on-demand price for the similar instance type in "us-west" data center is observed as 0.12$, which is very high as compared to the mean spot price. The size of duration varies according to the job size. Job size is the total time to execute without failure on a selected VM. It varies from 25 hours to 125 hours with a granularity of 25. The total cost of job execution depends on the number of checkpoints, as described in Equation 1. A job running on a spot instance is called successful or completed if the spot price of that instance remains under the user's bid price throughoutthe lifetime of the job. The following metrics are used to compare the effectiveness ofusing spot VMs over on-demand instances.

- **Reliability:** The reliability (R) for a job type can be defined as:

$$R = \frac{Number\ of\ successful\ jobs}{Total\ number\ of\ jobs}$$

- **Cost saving:** The cost saving for a job J_δ, started running on a VM type i at time t:

$$\text{Cost Saving} = \sum_{h=0}^{\delta} O_i\text{-}S_i(t+h) * F(t) \qquad (3)$$

Where O_i is the on-demand price of VM type i, where S_i is the spot price of VM type i, h is the time interval in hours, and F(t) is the status function that shows the availability of VMs. F (t) can be one if the VM is available; otherwise, zero.

Table 2 Spot Price information

Min Price($)	0.0140
Max Price($)	1.2600
Mean Price($)	0.0242
Bid Value($)	0.0152

Job Size vs. Number of Checkpoints required is plotted in Figure 5. In the graph, the *X* axis represents job size in hours, and the *Y* axis represents the number of checkpoints taken during the execution of jobs. It can be observed from the figure that the proposed KBCS requires less number of checkpoints in comparison to the number of checkpoints required in HCS and RECS.

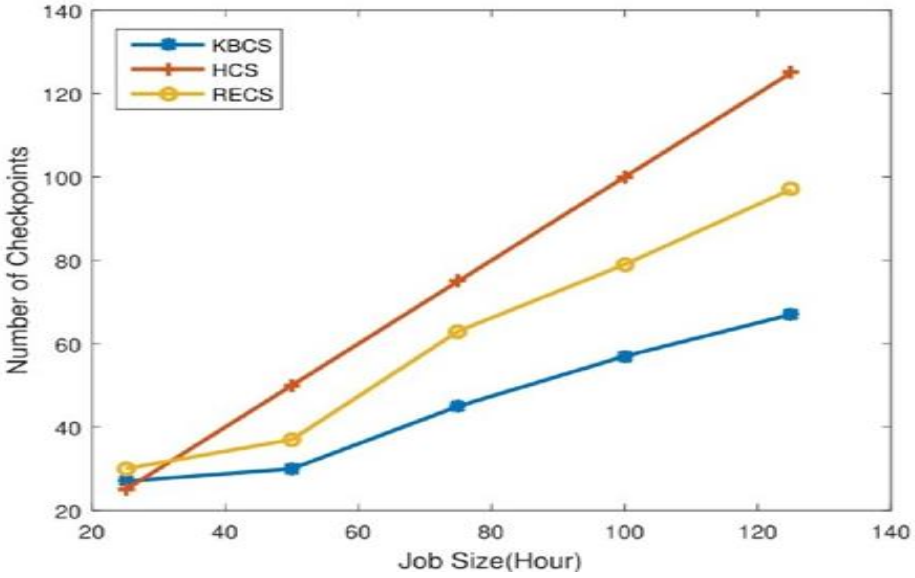

Figure 5 Job size vs. Number of checkpoints

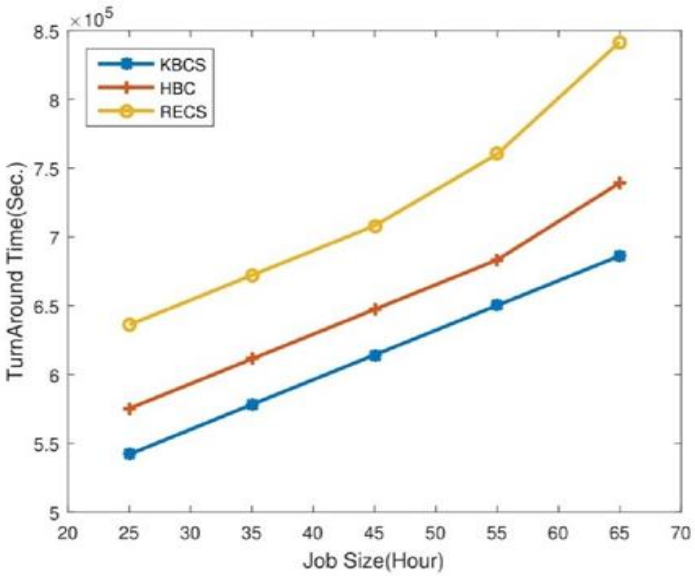

Figure 6 Job size vs. Turnaround time

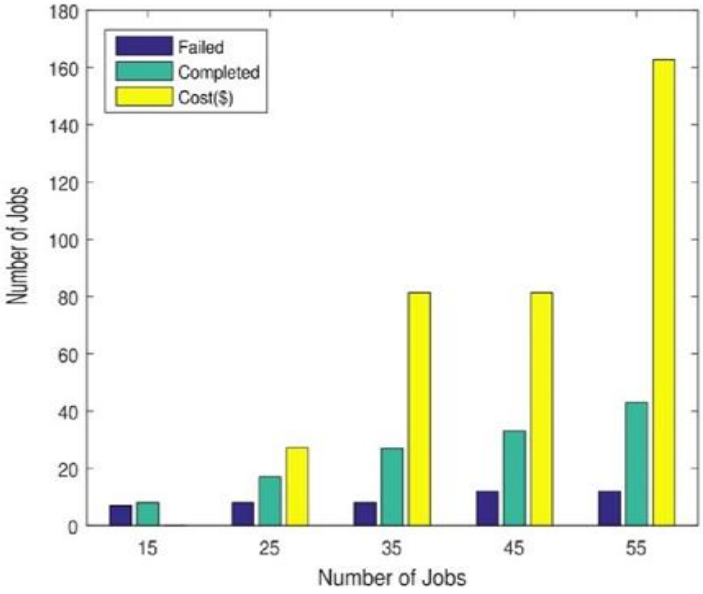

Figure 7 Impact of number of jobs on total cost & reliability

Figure 6 shows the Job Size vs. Turnaround time graph. The *graph's X-axis* represents the job size in hours, and *the* Y-axis represents the turnaround time in seconds. It signifies that KBCS outperforms HCS and RECS regarding turnaround time for a job size varying from 25 hours to 65 hours. Results show that KBCS results in 18% and 9% less turnaround time compared to the HCS and RECS, respectively. The difference between the turnaround time across the three strategies increases as the number of jobs varies from 25 to 65. Figure 7 shows the effect of the number of jobs on the total cost and reliabilityof the VM on which the job is executed. In the figure, *X* and *Y* axis both represent the number of jobs. Results show that even with a minimum bidding policy, onecan achieve 54% to 78% reliability with a bottom-line cost saving of 78% using spot instances, as the number of jobs varies from 15 to 55. As the number of jobs increases, the number of completed jobs increases rapidly compared to the failed jobs.

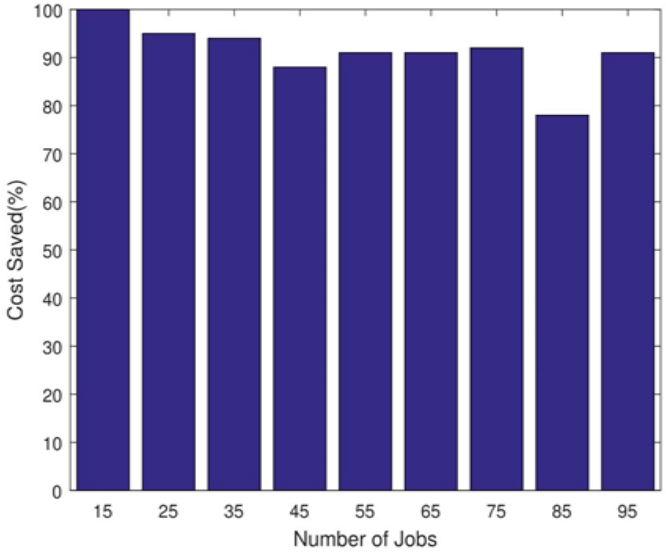

Figure 8 Number of jobs vs. Cost saved

Figure 8 depicts the cost saving for different job sizes on spot instances compared to the on-demand VMs. In the figure, X-axis represents the number of jobs, and Y axis represents the percentage of cost saved. The size of the job is fixed at 7 hours, and the number of jobs varies from 15 to 95 during the simulation.

7. Conclusion

This paper proposes a knowledge-based checkpointing strategy (KBCS) to minimize the overall checkpointing overhead. An algorithm for KBCS is presented. The correctness of the algorithm is verified. Results show that the proposed strategy outperforms the existing strategies (i.e., HCS and RECS) regarding overall checkpointing overhead within the job lifetime. It reduces the turnaround time by 18% compared to Hourly Checkpointing Strategy and 9% compared to Rising-Edge Checkpointing Strategy. One can also achieve 54% to 78% reliability with a cost saving of 78% for the workload used with the devised knowledge-based checkpointing algorithm. As part of our future work, we are examining the behavior of the proposed strategy across different types of instances and regions. We have also considered migration with our checkpointing approach to reduce job completion time further.

8. References

Agarwal, S., Mishra, A. K., & Yadav, D. K. (2017). Forecasting price of amazon spot instances using neural networks. *International Journal of Applied Engineering Research*, *12*(20), 10276-10283.

Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., & Tsafrir, D. (2013). Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation (TEAC)*, *1*(3), 1-20. https://doi.org/10.1145/2509413.2509416

Alourani, A., & Kshemkalyani, A. D. (2020, July). Provisioning spot instances without employing fault-tolerance mechanisms. In *2020 19th International Symposium on Parallel and Distributed Computing (ISPDC)* (pp. 126-133). IEEE. https://doi.org/10.1109/ISPDC51135.2020.00026

Amazon Web Services, Inc. (n.d.a). Secure and resizable cloud compute – Amazon EC2. Retrieved June 28, 2022, from https://aws.amazon.com/ec2/

Amazon Web Services, Inc. (n.d.b). Amazon EC2 Spot – Save up-to 90% on On-Demand Prices. Retrieved June 28, 2022, from https://aws.amazon.com/ec2/spot/

Amazon Web Services. (n.d.c). High-Performance Block Storage Retrieved June 27, 2022, from https://aws.amazon.com/ebs/

Andrzejak, A., Kondo, D., & Yi, S. (2010, August). Decision model for cloud computing under SLA constraints. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (pp. 257-266). IEEE. https://doi.org/10.1109/MASCOTS.2010.34

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599-616. https://doi.org/10.1016/j.future.2008.12.001

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, *41*(1), 23-50. https://doi.org/10.1002/spe.995

Cao, J., Simonin, M., Cooperman, G., & Morin, C. (2015). Checkpointing as a Service in Heterogeneous Cloud Environments. https://doi.org/10.1109/ccgrid.2015.160

Chen, C., Lee, B. S., & Tang, X. (2014, December). Improving hadoop monetary efficiency in the cloud using spot instances. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science* (pp. 312-319). IEEE. https://doi.org/10.1109/CloudCom.2014.35

Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A. N., & Krintz, C. (2010). See

spot run: using spot instances for mapreduce workflows. *HotCloud*, *10*, 1-7. https://dl.acm.org/doi/10.5555/1863103.1863110

Dawoud, W., Takouna, I., & Meinel, C. (2012, June). Increasing spot instances reliability using dynamic scalability. In *2012 IEEE Fifth International Conference on Cloud Computing* (pp. 959-961). IEEE. http://dx.doi.org/10.1109/CLOUD.2012.58

Deldari, A., & Salehan, A. (2021). A survey on preemptible IaaS cloud instances: challenges, issues, opportunities, and advantages. Iran Journal of Computer Science, 4(3), 1-24. https://doi.org/10.1007/s42044-020-00071-1

Di, S., Robert, Y., Vivien, F., Kondo, D., Wang, C. L., & Cappello, F. (2013, November). Optimization of cloud task processing with checkpoint-restart mechanism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (pp. 1-12). http://doi.acm.org/10.1145/2503210.2503217

Goiri, Í., Julia, F., Guitart, J., & Torres, J. (2010, April). Checkpoint-based fault-tolerant infrastructure for virtualized service providers. In *2010 IEEE network operations and management symposium-NOMS 2010* (pp. 455-462). IEEE. https://doi.org/10.1109/NOMS.2010.5488493

Google Cloud. (n.d). Compute Engine: Virtual Machines (VMs). Retrieved June 27, 2022, from https://cloud.google.com/compute/

Hussain, Z., Znati, T., & Melhem, R. (2019, May). Optimal placement of in-memory checkpoints under heterogeneous failure likelihoods. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 900-910). IEEE. https://doi.org/10.1109/IPDPS.2019.00098

IBM. (n.d.). Cloud Infrastructure Solutions. Retrieved June 25, 2022, from https://www.ibm.com/in-en/cloud/infrastructure

Jangjaimon, I., & Tzeng, N. F. (2015). Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. *IEEE Transactions on Computers*, *64*(2), 396-409. https://doi.org/10.1109/TC.2013.225

Jangjaimon, I., & Tzeng, N. F. (2013, May). Adaptive incremental checkpointing via delta compression for networked multicore systems. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing* (pp. 7-18). IEEE. http://dx.doi.org/10.1109/IPDPS.2013.33

Javadi, B., Thulasiram, R. K., & Buyya, R. (2013). Characterizing spot price dynamics in public cloud environments. *Future Generation Computer Systems*, *29*(4), 988-999. http://dx.doi.org/10.1016/j.future.2012.06.012

Javadi, B., Thulasiramy, R. K., & Buyya, R. (2011, December). Statistical modeling of spot instance prices in public cloud environments. In *2011 fourth IEEE international conference on utility and cloud computing* (pp. 219-228). IEEE. https://doi.org/10.1109/UCC.2011.37

Jung, D., Chin, S., Chung, K., Yu, H., & Gil, J. (2011). An efficient checkpointing scheme using price history of spot instances in cloud computing environment. In *Network and Parallel Computing: 8th IFIP International Conference, NPC 2011, Changsha, China, October 21-23, 2011. Proceedings 8* (pp. 185-200). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-24403-2_16

Liu, W., Wang, P., Meng, Y., Zhao, C., & Zhang, Z. (2020). Cloud spot instance price prediction using kNN regression. *Human-centric Computing and Information Sciences*, *10*(1), 1-14. https://doi.org/10.1186/s13673-020-00239-5

Mattess, M., Vecchiola, C., & Buyya, R. (2010, September). Managing peak loads by leasing cloud infrastructure services from a spot market. In *2010 IEEE 12th International Conference on High Performance Computing and*

Communications (HPCC) (pp. 180-188). IEEE. https://doi.org/10.1109/HPCC.2010.77

Mishra, A. K., Kesarwani, A., & Yadav, D. K. (2019a, March). Short term price prediction for preemptible vm instances in cloud computing. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)* (pp. 1-9). IEEE. https://doi.org/10.1109/I2CT45611.2019.9033677

Mishra, A. K., Umrao, B. K., & Yadav, D. K. (2018). A survey on optimal utilization of preemptible VM instances in cloud computing. *The Journal of Supercomputing*, *74*, 5980-6032. https://doi.org/10.1007/s11227-018-2509-0

Mishra, A. K., Yadav, D. K., Kumar, Y., & Jain, N. (2019b). Improving reliability and reducing cost of task execution on preemptible VM instances using machine learning approach. *The Journal of Supercomputing*, *75*, 2149-2180. https://doi.org/10.1007/s11227-018-2717-7

Popovici, F. I., & Wilkes, J. (2005, November). Profitable services in an uncertain world. In *SC'05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (pp. 36-36). IEEE. https://doi.org/10.1109/SC.2005.58

Ramesh, A., Pradhan, V., & Lamkuche, H. (2021, July). Understanding and analysing resource utilization, costing strategies and pricing models in cloud computing. In *Journal of Physics: Conference Series*, 1964(4), Article 042049. https://doi.org/10.1088/1742-6596/1964/4/042049

Sharma, P., Irwin, D. E., & Shenoy, P. J. (2016). How Not to Bid the Cloud. In A. Clements & T. Condie (Eds.), 8th {USENIX} Workshop on Hot Topics in Cloud Computing, HotCloud 2016, Denver, CO, USA, June 20-21, 2016. {USENIX} Association. https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/sharma

Song, Y., Zafer, M., & Lee, K. W. (2012, March). Optimal bidding in spot instance market. In *2012 Proceedings IEEE Infocom* (pp. 190-198). IEEE. https://doi.org/10.1109/INFCOM.2012.6195567

Toosi, A. N., Vanmechelen, K., Khodadadi, F., & Buyya, R. (2016). An auction mechanism for cloud spot markets. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, *11*(1), 1-33. https://doi.org/10.1145/2843945

Voorsluys, W., & Buyya, R. (2012, March). Reliable provisioning of spot instances for compute-intensive applications. In *2012 IEEE 26th international conference on advanced information networking and applications* (pp. 542-549). IEEE. https://doi.org/10.1109/AINA.2012.106

Wang, Y. M., Huang, Y., Vo, K. P., Chung, P. Y., & Kintala, C. (1995, June). Checkpointing and its applications. In *Twenty-fifth International Symposium on fault-tolerant Computing. Digest of papers* (pp. 22-31). IEEE. https://doi.org/10.1109/FTCS.1995.466999

Wu, L., Garg, S. K., & Buyya, R. (2012). SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. *Journal of Computer and System Sciences*, *78*(5), 1280-1299. https://doi.org/10.1016/j.jcss.2011.12.014

Yang, S., Khuller, S., Choudhary, S., Mitra, S., & Mahadik, K. (2021, December). Scheduling ML training on unreliable spot instances. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion* (Article 29, pp. 1-8). https://doi.org/10.1145/3492323.3495594

Yi, S., Andrzejak, A., & Kondo, D. (2012). Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Transactions on Services Computing*, *5*(4), 512-524. https://doi.org/10.1109/TSC.2011.44

Yi, S., Kondo, D., & Andrzejak, A. (2010, July). Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *2010 IEEE 3rd International Conference on Cloud Computing* (pp. 236-243). IEEE. https://doi.org/10.1109/CLOUD.2010.35