International Journal of Cloud Applications and Computing
Volume 12 • Issue 1

Cost-Effective Spot Instances Provisioning Using Features of Cloud Markets

Abdullah Alourani, University of Illinois at Chicago, Chicago, USA & Department of Computer Science and Information, College of Science in Zulfi, Majmaah University, Al-Majmaah, Saudi Arabia*

Ajay D. Kshemkalyani, Department of Computer Science, College of Engineering, University of Illinois at Chicago, Chicago, USA

ABSTRACT

Cloud computing offers a variable-cost payment scheme that allows cloud customers to specify the price they are willing to pay for renting spot instances at much lower costs than fixed payment schemes, and depending on the varying demand from cloud customers, cloud platforms could revoke spot instances at any time. To alleviate the effect of spot instance revocations, applications often employ different fault-tolerance mechanisms to minimize or even eliminate the lost work for each spot instance revocation. However, these fault-tolerance mechanisms incur additional overhead related to application completion time and deployment cost. This article proposes a novel cloud market-based approach for provisioning spot instances using features of cloud markets to reduce the deployment cost and completion time of applications. The simulation results show that the approach reduces the deployment cost and completion time compared to approaches based on fault-tolerance mechanisms.

KEYWORDS

Cloud Computing, Cloud Spot Market Features, Fault-Tolerance Mechanisms, Payment Schemes, Spot Instance Revocations, Spot Instances

INTRODUCTION

In this section, the authors describe cloud spot markets, discuss different types of fault-tolerance mechanisms, and present our major contributions.

Cloud Spot Markets

Cloud computing offers a variable-cost payment scheme that allows cloud customers to specify the price they are willing to pay for renting spot instances to run their applications at much lower costs than fixed payment schemes, and depending on the varying demand from cloud customers, cloud platforms could revoke spot instances at any time. The price of a spot instance can increase if the demand increases and the number of available instances that can be supported by a finite number

DOI: 10.4018/IJCAC.308276 *Corresponding Author

of physical resources in a data center of cloud providers decreases. Conversely, the price of this spot instance can decrease if the demand decreases and the number of available instances increases. Therefore, if the customer's price is greater than the cloud provider's price that depends on the demand, a spot instance will be provisioned to cloud customers' applications at the customer's price. However, when spot instances are already provisioned to cloud customer applications and the cloud provider's price goes above the customer's price, the cloud providers will terminate those spot instances within two minutes by sending termination notification signals. As a result, even though cloud customers sometimes rent spot instances at 90% lower prices than on-demand prices (Amazon, 2022), their applications that run on spot instances can be terminated based on price fluctuations that happen frequently; thus, those applications may incur additional overhead related to application completion time and deployment cost from re-executing lost work for each spot instance revocation.

Fault-Tolerance Mechanisms

Applications may benefit from different fault-tolerance mechanisms to alleviate the work lost for each spot instance revocation. However, these fault-tolerance mechanisms incur additional overhead related to application completion time and deployment cost. Fault-tolerance mechanisms are typically divided into three types: migration, checkpointing, and replication. First, migration mechanisms are often employed to reactively migrate the state of an application (i.e., memory and local disk state) to another instance prior to a spot instance revocation (Goundar et al., 2018), as illustrated in Figure 1. The overhead of a migration mechanism is determined based on the migration time of an application and the number of spot instance revocations during the application execution. The migration time of an application mostly depends on the resource usage of the application, whereas the number of spot instance revocations depends on the volatility of cloud spot markets (i.e., availability zones in AWS regions). A larger resource usage of an application often results in a higher overhead of a migration mechanism. Conversely, a smaller resource usage of an application often results in a lower overhead of a migration mechanism. A similar explanation is applicable for the volatility of cloud spot markets; thus, a higher overhead of a migration mechanism will lead to a higher overhead of an application's completion time and deployment cost.

Second, checkpointing mechanisms are often employed to proactively checkpoint an application's state to remote storage (e.g., AWS S3) (Jaswal et al., 2022), as illustrated in Figure 2. The overhead of a checkpointing mechanism is specified based on the time to checkpoint an application's state and the number of checkpoints, which represents how often an application's state is stored in remote

Figure 1. An overview of a migration mechanism

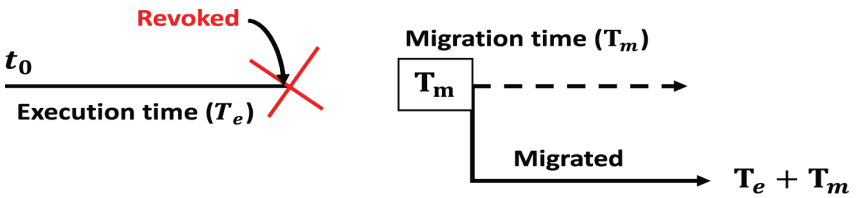
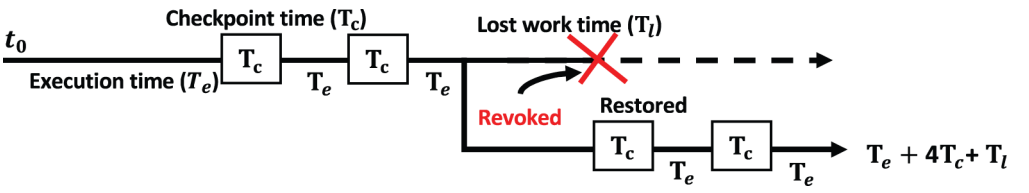


Figure 2. An overview of a checkpointing mechanism



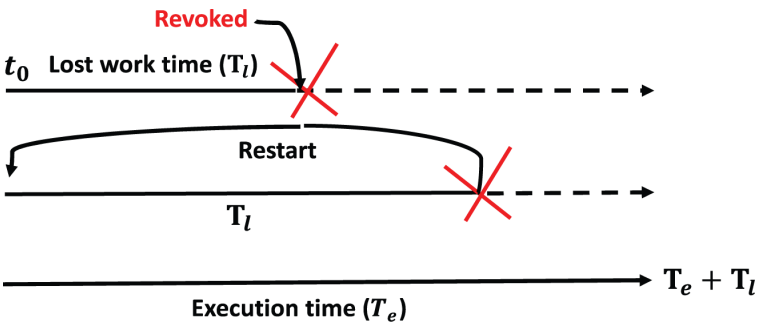
storage during the application execution, along with the time to re-execute the lost work from the last checkpoint for each spot instance revocation. The checkpointing time of an application relies on the resource usage of the application and the number of checkpoints typically specified by engineers who maintain applications deployed on spot instances. If engineers specify a large number of checkpoints, the overhead time to re-execute the lost work from the last checkpoint for each spot instance revocation will likely decrease, whereas the overhead time to checkpoint the state of an application will likely increase. Conversely, if engineers specify a small number of checkpoints, the overhead time to checkpoint the state of an application will likely decrease, whereas the overhead time to re-execute the lost work from the last checkpoint for each spot instance revocation will likely increase. Hence, checkpointing mechanisms require analyzing cloud spot markets and the resource usage of applications to optimize the tradeoff between the overhead of actual checkpoints and the overhead of re-executing lost work.

Third, replication mechanisms are often employed to replicate the computations of an application among different instances (Panda et al., 2018), as illustrated in Figure 3. The overhead of a replication mechanism is based on the degree of replication (i.e., the number of replicated instances) and the number of revocations that depends on the volatility of cloud spot markets and is independent of the resource usage of an application. As a result, a higher overhead of these fault-tolerance mechanisms leads to a higher overhead related to application completion time and deployment cost. The motivation behind this work is that spot instance revocations are rare in practice (Sharma et al., 2017). Although the availability of spot instances cannot be guaranteed, the authors provision a spot instance with a significantly large lifetime resulting in lower deployment costs compared to fault-tolerance mechanisms.

Major Contributions

The authors address a challenging problem for applications deployed on cloud spot instances that results from the overhead of employing fault-tolerance mechanisms. The authors propose a novel cloud market-based approach for Provisioning Spot Instances using FEatures of cloud Markets (P-SIFEM) to reduce the deployment cost and completion time of applications. P-SIFEM is composed of two key ideas. (1) A key idea is that the authors can eliminate the additional overhead resulting from employing fault-tolerance mechanisms by provisioning a spot instance with a high likelihood of completing jobs before revocation. (2) Another idea is that the authors can reduce consequent revocations when a spot instance is revoked by provisioning a new spot instance with the next highest lifetime and a high revocation gap with the revoked spot instance. The authors evaluate P-SIFEM in simulations and use Amazon spot instances that contain jobs in Docker containers and realistic price traces from EC2 markets. Our simulation results show that our approach reduces the deployment cost and completion time compared to approaches based on fault-tolerance mechanisms. The P-SIFEM code and our simulation results are publicly available.

Figure 3. An overview of a replication mechanism



This paper is an extension of our paper with the following new contributions:

- The authors present additional experimental results for evaluating the effectiveness of P-SIFEM with different settings of a fault-tolerance approach, along with an illustrative example of P-SIFEM.
- The authors add a formal framework, model, and mathematical description for the overall deployment time and cost when using P-SIFEM and the fault-tolerance approach.
- The authors add a formal framework, model, and mathematical description for features of cloud spot markets used by P-SIFEM, such as the spot instance lifetime, revocation probability, and revocation gap between cloud spot markets.
- The authors provide a detailed literature review and threats to the validity of P-SIFEM, along with multiple directions for future work.

The remainder of this paper is organized as follows. The related work section presents related work. The problem statement section defines the problem statement. In our approach section, the authors explain our approach for Provisioning Spot Instances using FEatures of cloud Markets (P-SIFEM), describe our key ideas for P-SIFEM, and explain the P-SIFEM algorithm. The evaluation section provides the evaluation. The results section discusses the results. Finally, the authors conclude this work in the conclusion section.

RELATED WORK

In this section, the authors discuss the related work concerning modeling spot markets, employing fault-tolerance mechanisms, and optimizing resource provisioning.

Modeling Spot Markets

There is a large body of work in modeling spot markets to reduce the spot instance cost and the performance penalty that results from a high number of revocations by designing optimal bidding strategies (Khodak et al., 2018; Herzfeldt et al., 2020) and developing prediction schemes (Mishra et al., 2019). Khodak et al. (2018) proposed an adaptive bidding approach that leverages cloud dynamics to optimize spot instance bidding strategies. Mishra et al. (2019) proposed an approach based on probability between historical price transitions for short term price prediction of spot instances. Javadi et al. (2011) proposed a statistical approach to analyze changes in spot price variations and the time between price variations to explore the characterization of spot instances that are required to design fault-tolerant algorithms for applications deployed on cloud spot instances.

Employing Fault-Tolerance Mechanisms

Several prior works focused on reducing the effect of spot instance revocations using fault-tolerance methods (Subramanya et al., 2015; Goundar et al., 2018), such as virtual machine (VM) migration (Sharma et al., 2017; Singh et al., 2022; Shastri et al., 2017; Soltani et al., 2020; Priyanka et al., 2021), replication (Dharwadkar et al., 2018; Panda et al., 2018), and checkpointing (Jaswal et al., 2022; Subramanya et al., 2015). Subramanya et al. (2015) proposed a batch computing service that chooses a fault-tolerance approach and a cloud spot market to reduce the effect of spot instance revocations without requiring application change. Sharma et al. (2017) focused on changing applications to determine and migrate to the lowest cost instances, leading to large deployment cost savings with a neglectable impact on completion time. However, this work is subject to limited types of application architectures, such as MapReduce architectures not being applicable to changing applications, as spot prices change due to geographical constraints. Shastri et al. (2017) proposed a resource container that enables applications to self-migrate to new spot VMs in a way that optimizes cost efficiency as

cloud spot prices change. Dharwadkar et al. (2018) proposed an task scheduling approach based on replication strategies to economically and efficiently run scientific applications. Jaswal et al. (2022) proposed a checkpoint mechanism that effectively identifies malicious faults to ensure the reliability of provided cloud services.

Optimizing Resource Provisioning

There has been significant prior work on optimizing resource provisioning, including payment schemes (Vinothina et al., 2022; Bisht et al., 2022; Aliyu et al., 2020), resource elasticity (Ahuja et al., 2020; Nandal et al., 2021; Sahana et al., 2020; Gond et al., 2019; Youssef et al., 2018) resource reclamations (Sharma et al., 2019; Funaro et al., 2019; Harrath et al., 2019), testing the effect of spot instance revocations (Alourani et al., 2020), and optimizing performance (Kapgate, 2021; Ahammad et al., 2021; Swarnakar et al., 2021). Sharma et al. (2019) presented a hybrid approach based on pricing and system models that encourages self-capping to increase cloud utilization. Aliyu et al. (2020) presented a hybrid meta-heuristic approach based on the ant colony optimization model to effectively provision cloud computing resources. Bisht et al. (2022) proposed a workflow scheduling algorithm based on Resources from heterogeneous environments to reduce makespan, energy consumption, load balancing, and deployment cost. Harrath et al. (2019) proposed a resource provisioning approach based on a multi-objective genetic algorithm to provision real-time tasks to cloud computing resources.

Critical Analysis

To the best of our knowledge, P-SIFEM is the first solution for provisioning spot instances without employing fault-tolerance mechanisms. While many of the prior works focused on reducing the effect of spot instance revocations by modeling spot markets and using fault-tolerance methods, these works are subject to altering pricing algorithms and are exposed to incurring overhead related to application completion time and deployment cost, respectively. In contrast, P-SIFEM leverages features of cloud spot markets to mitigate the effect of spot instance revocations. Additionally, P-SIFEM is orthogonal to optimizing resource provisioning approaches to economically deploy applications in clouds.

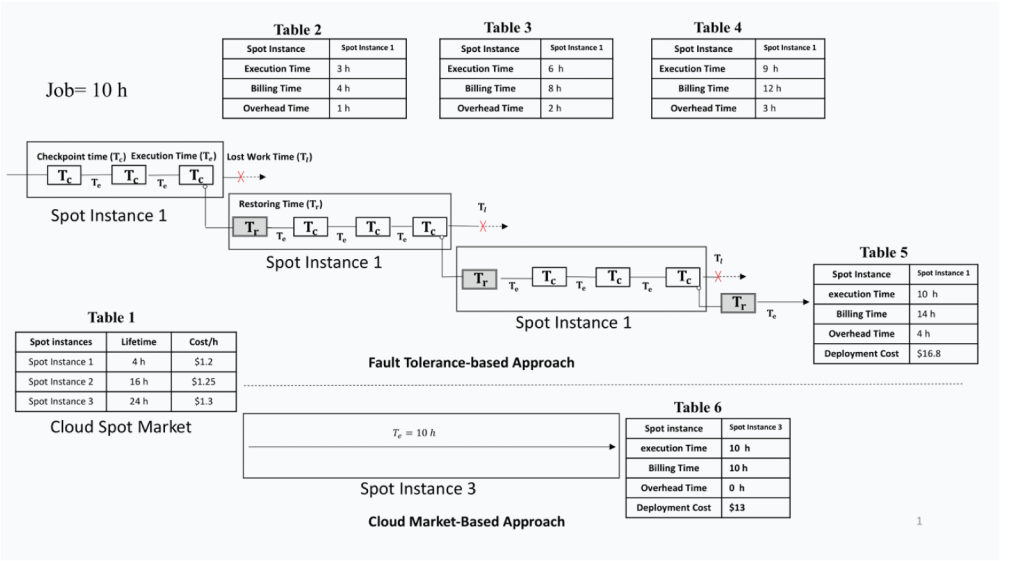
PROBLEM STATEMENT

In this section, the authors describe an illustrative example of P-SIFEM and formulate the problem statement.

An Illustrative Example

An illustrative example is shown in Figure 4. Applications deployed on cloud spot instances are often exposed to revocations by cloud providers, and as a result, these applications often employ various fault-tolerance mechanisms to alleviate the effect of spot instance revocations. However, these fault-tolerance mechanisms often incur additional overhead related to application completion time and deployment cost. Our illustrative example shows a comparison of deployment costs for provisioning spot instances using a fault-tolerance approach and a cloud market-based approach (i.e., P-SIFEM). Since cloud spot instances are often used to run batch job applications, the authors use a batch job application throughout the illustrative example to compute the deployment cost for provisioning spot instances using these approaches. As an example, the authors assume a cloud spot market contains three spot instances (i.e., Spot Instance 1, Spot Instance 2, and Spot Instance 3) that meet the resource requirements for a batch job (i.e., a job of 10 hours execution length and 64 GB of memory footprint). For ease of calculation, the authors assign a fixed price per hour for each spot instance throughout the entire job runtime. The prices of Spot Instance 1, Spot Instance 2, and Spot Instance 3 are $1.2, $1.25, and $1.3, respectively, and the lifetimes of Spot Instance 1, Spot Instance 2, and Spot Instance 3 are 4, 16, and 24 hours, respectively, as illustrated in Table 1 in Figure 4.

Figure 4. An illustrative example of P-SIFEM. T_c designates the checkpoint time, T_e designates the execution time, T_l designates the lost work time, and T_r designates the restoring time. The overhead time includes the checkpoint time, the restoring time, and the re-execution time of the lost work. Table 1 includes an overview of cloud spot markets: the names of spot instances followed by their lifetimes and their costs per hour rate (i.e., a single billing cycle in cloud platforms (Amazon, 2022)). The other tables represent the deployment information at certain points of execution.



First, the authors run the job using a fault-tolerance approach that employs a checkpointing mechanism and a cost-driven selection policy that selects a spot instance with the lowest price. To employ the checkpointing mechanism, the authors need to specify the number of checkpoints in a way that balances the overhead of actual checkpointing and the overhead of re-executing the lost work from the last checkpoint for each spot instance revocation. Since the deployment cost depends on the number of billing cycles, the authors specify the number of checkpoints for a job based on the number of billing cycles (i.e., a checkpoint is taken in each billing cycle). Suppose the time to checkpoint the state of a job to remote storage (i.e., T_c) is five minutes and the time to restore a checkpoint from remote storage (i.e., recovery time or T_r) is also five minutes. Initially, Spot Instance 1 will be selected based on the cost-driven selection policy to run the job (i.e., T_e) until Spot Instance 1 is revoked after four hours according to the lifetime of Spot Instance 1. Additionally, a checkpoint will be taken/stored in each billing cycle (i.e., an hour based on the billing policies of various cloud computing platforms (Amazon, 2022)). Spot Instance 1 will complete executing three hours of the job and 15 minutes for storing three checkpoints before Spot Instance 1 is revoked at its fourth hour of execution according to the lifetime of Spot Instance 1, and there will be 45 minutes of lost work that was executed but not saved into remote storage (i.e., a checkpoint). Thus, the billing time is four hours, whereas the completed execution time of the job is three hours and the overhead time resulting from checkpoints and lost work is one hour, as illustrated in Table 2 in Figure 4. To resume the job execution, Spot Instance 1 will again be selected based on the cost-driven selection policy; then, the last checkpoint will be restored, which takes five minutes, to resume the execution for another three hours plus 15 minutes for storing three checkpoints before this spot instance is revoked at its fourth hour of execution, and there will be 40 minutes of lost work that was executed but not saved in remote storage. Thus, the billing time increases by four hours to become eight hours, whereas the completed execution time of the job increases by three

hours to become six hours in total and the overhead time increases by one hour to become two hours in total, as illustrated in Table 3 in Figure 4. Similarly, the next run will complete executing another three hours, 20 minutes for storing/restoring checkpoints, and 40 minutes of lost work. At this point, the billing time is 12 hours, whereas the completed execution time of the job is nine hours, and the overhead time is three hours, as illustrated in Table 4 in Figure 4. Again, Spot Instance 1 will be selected, and the last checkpoint will be restored to resume the remaining execution of the job for the last hour; then, Spot Instance 1 will be revoked due to the completion of the job execution. Since the last execution time is one hour and five minutes, the billing time will be rounded up to two hours based on the billing policy that charges are counted per billing cycle (i.e., a complete hour). The billing time is 14 hours, whereas the completed execution time of the job is 10 hours, and the overhead time is four hours, as illustrated in Table 5 in Figure 4. As a result, the total cost of executing this job using the fault-tolerance approach will be $16.8, which is the billing time (i.e., 14 hours) multiplied by the price of selected spot instances throughout the entire job runtime (i.e., the price of Spot Instance 1, which is $1.2).

Second, the authors run the job using a cloud market-based approach that uses the spot instance lifetime and a lifetime-driven selection policy that selects the spot instance with the highest lifetime. To reduce the revocation risk of this policy, the authors limit the selection of spot instances to instances whose lifetimes are significantly higher than the job's execution length. When using the cloud market-based approach, if a spot instance is revoked, the job will be re-executed from the beginning, and the work before the revocation will be lost. When the job is executed using the cloud market-based approach, Spot Instance 3 will be selected based on the lifetime-driven selection policy to execute the job until the job execution is completed or Spot Instance 3 is revoked after 24 hours according to the lifetime of Spot Instance 3. Spot Instance 3 will complete 10 hours of the job execution and will be terminated before it is revoked according to the lifetime of Spot Instance 3, as illustrated in Table 6 in Figure 4. Thus, the total cost of executing this job using the cloud market-based approach will be $13, which is the billing time (i.e., 10 hours) multiplied by the price of selected spot instances throughout the entire job runtime (i.e., the price of Spot Instance 3, which is $1.3). In summary, even though the fault-tolerance approach selects the most inexpensive spot instance in the cloud spot market to run the job, this approach leads to a higher deployment cost resulting from the overhead of the fault-tolerance approach (i.e., the checkpointing mechanism). On the other hand, the cloud market-based approach selects the most expensive spot instance in the cloud spot market but results in a lower deployment cost since this approach does not incur any additional overhead resulting from employing fault-tolerance mechanisms. As a result, the cloud market-based approach enables cloud customers to avoid unnecessary overhead resulting from employing fault-tolerance mechanisms while benefiting from spot instances' extraordinarily lower prices compared to on-demand instances' prices.

Finally, although cloud market-based approaches could reduce deployment costs compared to fault-tolerance approaches, cloud market-based approaches cannot guarantee the availability of spot instances that depend on the varying demands of many cloud customers, leading to higher deployment costs and completion times than the fault-tolerance approaches. For example, when the job is executed using the cloud market-based approach, Spot Instance 3 will be selected based on the lifetime-driven selection policy to execute the job until the job execution is completed or Spot Instance 3 is revoked after 24 hours according to the lifetime of Spot Instance 3. However, suppose that Spot Instance 3 is revoked right before the completion of the job execution (e.g., after 9 hours) as a part of the normal behavior of spot instances. Then, the job will be re-executed from the beginning, and the work before the revocation will be lost. To resume job execution, Spot Instance 3 will again be selected based on the lifetime-driven selection policy. Spot Instance 3 will complete 10 hours of job execution and will be terminated before it is revoked according to its lifetime. Thus, the billing time increases by 10 hours to become 20 hours, and the total cost of executing this job using the cloud market-based approach will be $26, which is the billing time

(i.e., 20 hours) multiplied by the price of selected spot instances throughout the entire job runtime (i.e., the price of Spot Instance 3, which is $1.3). In this case, the cloud market-based approach leads to much higher deployment costs and completion time than the fault-tolerance approach. As a result, the volatility of cloud markets has a high impact on the effectiveness of cloud market-based approaches.

The Problem Statement

Cloud computing offers a variable-cost payment scheme that allows cloud customers to specify the price they are willing to pay for renting spot instances to run their applications at much lower costs than fixed payment schemes. In exchange, applications deployed on spot instances are often exposed to revocations by cloud providers, and as a result, these applications often employ different fault-tolerance mechanisms to minimize or even eliminate the lost work for each spot instance revocation. However, these fault-tolerance mechanisms incur additional overhead related to application completion time and deployment cost. In this paper, the authors address a challenging problem for applications deployed on cloud spot instances that results from the overhead of employing fault-tolerance mechanisms—determining how to effectively deploy applications on spot instances using features of cloud markets to reduce the deployment cost and completion time of applications. The root of this problem is that applications often employ fault-tolerance mechanisms to minimize the lost work for each spot instance revocation without taking into consideration the overhead of fault-tolerance mechanisms, leading to significantly larger deployment costs and completion times of applications, and as a result, the advantages of cloud spot instances could be significantly minimized or even completely eliminated:

$$m_s = \frac{\sum_{i=2}^{n} t_i^s - t_{i-1}^s}{n-1} \tag{1}$$

where n designates the total number of revocations of a spot instance s during the execution of a certain job. t_i^s and t_{i-1}^s are the revocation times of a spot instance s at times i and $i-1$, respectively, during job execution:

$$p_{j,s} = \frac{l_j}{m_s} \tag{2}$$

where j designates a certain job, l_j is job j's execution length, and m_s is the lifetime of spot instance s, as described by Eq. (1):

$$g_{s1,s2} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\left|t_i^{s1} - t_j^{s1}\right|}{n * m} \tag{3}$$

where n designates the total number of revocations for a spot instance s_1 and m designates the total number of revocations for a spot instance s_2 during the execution of a certain job. t_i^{s1} and t_j^{s2} designate the revocation time of a spot instance s_1 at time i and the revocation time of a spot instance s_2 at a corresponding time j, respectively, during the job execution.

OUR APPROACH

In this section, the authors state our key ideas for P-SIFEM, outline the architecture of P-SIFEM, and explain the P-SIFEM algorithm.

Key Ideas

A goal of our approach is to automatically provision spot instances without employing fault-tolerance mechanisms to reduce the deployment cost and completion time of applications. Our approach leverages features of cloud spot markets such as the spot instance lifetime, revocation probability, and revocation correlation between cloud spot markets and provision spot instances for applications. The spot instance lifetime (i.e., mean time to revocation (MTTR)) described by Eq. (1) represents the average time until a spot instance's price rises above the corresponding on-demand instance price because cloud customers are often not willing to pay more than the on-demand price to rent spot instances. The authors use the corresponding on-demand prices instead of customer bids to compute the lifetime of spot instances. Since Amazon recently changed its pricing policy (New Spot Instance Pricing, 2022), such that customers are no longer required to place bids, and since spot prices are based on supply and demand, revocations are no longer correlated to customer bids. The revocation probability of each spot instance described by Eq. (2) represents the estimated lifetime of a spot instance during a job execution and is calculated by dividing the job's execution length by the lifetime of the provisioned spot instance. The revocation correlation between cloud spot instances described by Eq. (3) represents how often these spot instances were revoked at the same time (i.e., the same hour representing a single billing cycle in cloud platforms (Amazon, 2022)) over three months from June 2019 to September 2019.

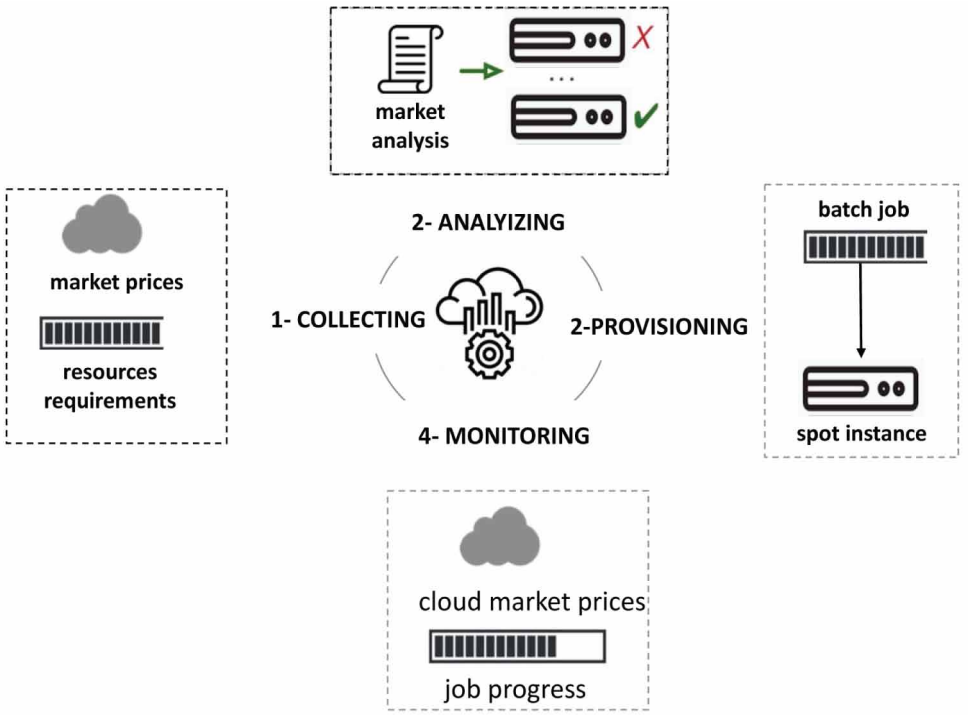
In general, cloud spot markets show a broad range of characteristics. These important characteristics are at the core of our approach. First, revocations rarely occur in some cloud spot markets, so the lifetime of these markets is very high (i.e., $>$ 600 hours) (Sharma et al., 2017). Although the availability of spot instances cannot be guaranteed, the authors provision a spot instance with a significantly large lifetime, resulting in lower deployment costs compared to fault-tolerance mechanisms. Second, employing fault-tolerance mechanisms often results in additional overhead related to application completion time and deployment cost (Subramanya et al., 2015). Third, cloud spot markets exhibit variations in price characteristics for a similar type of spot instance across various cloud spot markets (i.e., availability zones in AWS regions). Thus, a spot instance in a cloud market is often independent of a spot instance in another cloud market, which suggests that a spot instance's revocation in a cloud market is often uncorrelated with a spot instance in another cloud market (Sharma et al., 2017). Based on these characteristics, our key idea is that the authors can eliminate the additional overhead resulting from employing fault-tolerance mechanisms by provisioning a spot instance with a high likelihood of completing jobs before revocation.

Another idea is that the authors can reduce consequent revocations when a spot instance is revoked by provisioning a new spot instance with the next highest lifetime and a high revocation gap with the revoked spot instance. When the authors provision a spot instance that is uncorrelated with the revoked spot instance, it is more unlikely that the new spot instance will be revoked again than another spot instance that is highly correlated with the revoked spot instance. As a result, these key ideas enable cloud customers to avoid unnecessary overhead resulting from employing fault-tolerance mechanisms; hence, cloud customers can execute jobs with a completion time near that of on-demand instances but at a cost of only spot instances.

Overview of P-SIFEM

The architecture of P-SIFEM is illustrated in Figure 5. Cloud market features are at the core of P-SIFEM to provision spot instances for applications. Provisioning spot instances for applications based on cloud market features reduces the deployment cost of jobs compared to the deployment cost of jobs

Figure 5. The architecture of P-SIFEM



using a fault-tolerance approach or on-demand instances, in addition to maintaining a completion time near that of on-demand instances. There are four main phases in P-SIFEM. 1) Collecting cloud market prices and the resource requirements for a job. Initially, P-SIFEM uses EC2's REST API to collect cloud market prices for all instances (i.e., servers) across all markets (i.e., availability zones and regions) over three months. P-SIFEM supports a predefined resource usage of a job to guide the selection of spot instances and assumes a job's resource usage does not change significantly (i.e., unphased jobs) over runtime. 2) Analyzing cloud spot market features to identify a suitable spot instance for a job. P-SIFEM first filters cloud spot markets to identify spot instances that satisfy the job's resource usage requirements and then computes the lifetime for each spot instance, the revocation probability for the job and a certain spot instance, and the revocation correlation between cloud spot instances. P-SIFEM sorts the spot instances' lifetimes in descending order to provision the spot instance with the highest lifetime as long as the lifetime of the spot instance is significantly higher than the job's execution length. P-SIFEM uses the revocation probability to determine when a spot instance might be revoked during its execution.

Additionally, P-SIFEM uses the revocation correlation between a pair of cloud spot instances when the provisioned spot instance is revoked to provision a new spot instance that is less correlated or even uncorrelated with the revoked spot instance to reduce the likelihood that the new spot instance will again be revoked over the job's runtime. 3) Provisioning a suitable spot instance for the job. P-SIFEM uses the features of cloud spot markets and the resource requirements of spot instances to provision a suitable spot instance for a job. 4) Monitoring cloud market prices and the job execution progress over the job's execution. P-SIFEM monitors cloud market prices to determine when a spot instance is revoked based on the revocation probability of the provisioned spot instance. When the provisioned spot instance is revoked, P-SIFEM provides a new spot instance with the next highest lifetime and a low revocation correlation with the revoked spot instance. P-SIFEM also monitors the

progress of the job execution to determine when the job execution is completed. Finally, our hypothesis is that leveraging cloud market features without employing fault-tolerance mechanisms to provision spot instances for applications reduces the deployment cost compared to the deployment cost using fault-tolerance approaches or on-demand instances and maintains the completion time near that of on-demand instances.

Algorithm 1. P-SIFEM's algorithm for provisioning spot instances using features of cloud markets

```
1: Inputs: Jobs J, Cloud Markets M, Resources R
2: U ← FindSuitableServers(J, R)
3: L ← ComputeLifeTime(M, U)
4: for each j in J do
5:          S_j ← ServerBasedLifeTime (j, M, L)
6:          while j ← Completed do
7:                    s_j ← Highest(S_j)
8:                    if length(s_j) >> length(j) then
9:                              v_sj ← RevocationProbability (j, s_j)
10:                             ProvisionHighestLifeTime (j, s_j)
11:                             if s_j encounters v_sj then
12:                                       C_j, T_j ← C_j ∪ {c_sj}, T_j ∪ {t_sj}
13:                                       G_sj ← FindLowCorrelation (j, s_j)
14:                                       S_j ← (S_j \ {s_j}) ∩ G_sj
15:                             end if
16:                   end if
17:         end while
18:         C_j, T_j ← C_j ∪ {c_sj}, T_j ∪ {t_sj}
19:         C, T ← ComputeCostExeTime (C_j, T_j)
20: end for
21: return C, T
```

P-SIFEM Algorithm

P-SIFEM is illustrated in Algorithm 1 that takes in the batch job set J; the resource requirement set R; and the entire set of cloud markets M, containing on-demand instance types, prices of on-demand instances, spot instance types, their availability zones, their regions, and spot instance prices over three months. Starting from Step 2, the algorithm finds a suitable set of spot instances U that meet the resource requirements, which are provided by engineers who create and maintain cloud-based applications (e.g., batch jobs). In P-SIFEM, the authors use the memory size to determine suitable sizes of spot instances that are supported by EC2 markets (Amazon, 2022). The authors use the memory size to determine suitable spot instances, as the memory maintains the state of a running application, which has a significant influence on the overhead related to application completion time (i.e., the application's checkpointing time). P-SIFEM selects spot instances that exactly match the required size of the memory to ensure that application completion time does not vary when a larger memory size is permitted. In Step 3, for each suitable spot instance, the spot instance lifetime (i.e.,

the spot instance's MTTR) is computed based on the corresponding on-demand instance price. L is the set of such lifetimes.

In Steps 4-20, for each job, the algorithm is executed until the jobs in the job set are completed. In Step 5, the cloud spot markets are first filtered to include only a set of suitable spot instances S_j for the job j according to their lifetimes L, and then these spot instances are sorted in descending order based on their lifetimes. In Steps 6–17, job j is executed until the job's execution is completed. In Step 7, the algorithm selects a spot instance s_j with the highest lifetime. In Step 8, the authors ensure that the highest lifetime for the spot instance s_j is significantly higher than the job $j's$ execution length to reduce the revocation probability of the provisioned spot instance during the job execution. In Step 9, the algorithm computes the revocation probability of job j scheduled on instance s, called v_{sj}, by dividing the job $j's$ execution length by the lifetime of the provisioned spot instance s_j, as described by Eq. (2). In Step 10, the spot instance s_j with the highest lifetime is provisioned to (re)start executing job j:

$$T_P = \sum_{s=1}^{n} \left(st_s + et_s + rt_s \right) \tag{4}$$

where n designates the total number of provisioned spot instances to execute a certain job. st_s, et_s, and rt_s are the startup time, the execution time, and the re-execution time, respectively, of a provisioned spot instance s during the job execution when using P-SIFEM:

$$C_P = \sum_{s=1}^{n} \left(st_s + et_s + rt_s + bt_s \right) * p_s \tag{5}$$

where n designates the total number of provisioned spot instances to execute a certain job. st_s, et_s, rt_s, bt_s, and p_s are the startup time, the execution time, the re-execution time, the buffer time, and the price, respectively, of a provisioned spot instance s during job execution when using P-SIFEM.

In Steps 11–15, the algorithm checks whether the provisioned spot instance s_j is revoked based on its revocation probability v_{sj} during job execution j. When a spot instance s_j is revoked, the deployment time t_{sj} and cost c_{sj} are added to the total deployment time set T_j and cost set C_j, respectively, in Step 12. In P-SIFEM, the deployment time represents the job's execution time until the spot instance is revoked, the deployment cost of a spot instance represents the price of the provisioned spot instance at a certain execution point, and the cost is computed at a per hour rate (i.e., a single billing cycle in cloud platforms (Amazon, 2022)). In Step 13, the high revocation gap set G_{sj} with the revoked spot instance is computed using the revocation gap between cloud spot instances. In Step 14, the revoked spot instance is removed from the set of suitable spot instances S_j, and the set of suitable spot instances S_j is filtered based on a high revocation gap set G_{sj}. The cycle of Steps 6–17 repeats until the job $j's$ execution is completed. When the job $j's$ execution is completed, the deployment time t_{sj} and cost c_{sj} are added to the total deployment time set T_j and cost set C_j, respectively, in Step 18. In Step 19, the total deployment time set T_j and cost set C_j are computed and then added to the overall deployment time T (i.e., T_p) described by Eq. (4) and

cost C (i.e., C_p) described by Eq. (5), respectively. The cycle of Steps 4–20 repeats until the jobs in the job set are completed. Finally, the total deployment time T and cost C are returned in Step 21 as the algorithm ends.

EVALUATION

In this section, the authors describe the design of the study to evaluate P-SIFEM and state threats to its validity. The authors pose the following research questions (**RQs**):

RQ1: How efficient is P-SIFEM compared to a fault-tolerance approach in executing applications?
RQ2: How effective is P-SIFEM compared to a fault-tolerance approach in reducing the deployment cost of applications?
RQ3: Does optimizing the settings of a fault-tolerance approach eliminate the effectiveness of P-SIFEM?

Subject Applications

The authors evaluate P-SIFEM in simulations and use Amazon spot instances that contain jobs in Docker containers and realistic price traces from EC2 markets, which contain approximately 7,600 independent spot prices for different types of instances among 44 availability zones (i.e., data centers) in 16 regions. P-SIFEM packages jobs in Docker containers helps to simplify restoration and checkpointing. The authors use Amazon spot instances since their lifetimes often exceed hundreds of hours, unlike the lifetimes for Google preemptible instances, which are less than 24 hours (Sharma et al., 2019). Additionally, the authors use Docker containers since they support checkpointing and restoring container images. The authors use a load generator called Lookbusy (Carraway, 2022) to create synthetic jobs with different amounts of resource usage. In addition, P-SIFEM uses EC2's REST API to collect realistic price traces for all spot instances across all markets (i.e., availability zones and regions) for three months from June 2019 to September 2019.

The authors conduct some analysis on the collected cloud market prices to compute a spot instance's lifetime to identify the spot instance's lifetime based on its revocations over three months and to seed our P-SIFEM for provisioning spot instances (i.e., P-SIFEM looks for the spot instance with the highest lifetime to provision it for a job as long as the lifetime of this spot instance is significantly higher than the job's execution length). The authors also use the collected cloud market prices to compute the revocation correlation between cloud spot instances to identify how often a pair of spot instances were revoked at the same time (i.e., the same hour representing a single billing cycle (Amazon, 2022)) over three months and to seed our P-SIFEM for reprovisioning spot instances, i.e., P-SIFEM looks for a spot instance that has a low revocation correlation with the revoked spot instance to reduce the revocation probability of the provisioned spot instance over the job's execution. In other words, when the authors provision a spot instance that is less correlated with the revoked spot instance, it is more unlikely that the new spot instance will be revoked again than another spot instance that is highly correlated with the revoked spot instance. As a result, our P-SIFEM simulator utilizes these analyses of cloud markets to (re)provision spot instances without employing fault-tolerance mechanisms and hence reduces the deployment cost and completion time of applications:

$$T_F = \sum_{s=1}^{n} \left(st_s + et_s + vt_s + ct_s + rt_s \right) \tag{6}$$

where n designates the total number of provisioned spot instances to execute a certain job. st_s, et_s, vt_s, ct_s, and rt_s are the startup time, the execution time, the recovery time, the checkpointing time, and the re-execution time, respectively, of a provisioned spot instance s during job execution when using the checkpointing-based approach:

$$C_F = \sum_{s=1}^{n} \left(st_s + et_s + vt_s + ct_s + rt_s + bt_s \right) * p_s \tag{7}$$

where n designates the total number of provisioned spot instances to execute a certain job. st_s, et_s, vt_s, ct_s, rt_s, bt_s, and p_s are the startup time, the execution time, the recovery time, the checkpointing time, the re-execution time, the buffer time of billing cycles, and the price, respectively, of a provisioned spot instance s during the job execution when using the checkpointing-based approach.

The authors use a checkpointing-based fault-tolerance approach as the baseline. The authors set the frequency of checkpoints to be in every billing cycle (i.e., each hour) based on job length. The authors use different numbers of checkpoints during a job's execution to identify an optimal setting of this checkpointing-based approach. Moreover, the authors compute the total deployment time (i.e., T_F) and cost (i.e., C_F) for the checkpointing-based approach, as described by Eqs. (6) and (7), respectively.

Methodology

Some objectives of the experiments are to demonstrate that P-SIFEM can efficiently execute applications and can effectively decrease the deployment cost of applications compared to a fault-tolerance approach. For these objectives, the authors use different combinations of job execution length, job memory footprint, and number of revocations to show the impact on the completion time and the deployment cost when a spot instance is provisioned for the job using P-SIFEM and the fault-tolerance approach. The job execution lengths vary between 13 and 101 hours. The job memory footprints vary between 4 and 64 GB. The number of revocations per day of the job's execution length varies between 1 and 16 times. The authors define two revocation rules with different ranges for P-SIFEM and the fault-tolerance approach to show the impact on the completion time and the deployment cost for different numbers of revocations during a job's execution. When a spot instance is provisioned for a job using the fault-tolerance approach, the authors randomly send a fixed number of revocations per day of the job's execution length, as suggested by prior work (Subramanya et al., 2015). Conversely, when a spot instance is provisioned for a job using P-SIFEM, the authors use the revocation probability of a spot instance that relies on realistic price traces from the Amazon cloud to revoke the provisioned spot instance. The deployment cost/completion time for P-SIFEM is derived from the price/execution time of spot instances during the startup of a spot instance, the job's execution, and the job's re-execution after the provisioned spot instance is revoked. On the other hand, the deployment cost/completion time for the fault-tolerance approach is derived from the price/execution time of spot instances during the startup of a spot instance, the job's execution, the job's re-execution, the job's checkpointing, and the job's recovery (i.e., check point restoration). Evaluating P-SIFEM with different combinations of job settings (i.e., job execution length and job memory footprint) enables us to answer RQ1 and RQ2.

Another objective is to determine whether optimizing the settings of the fault-tolerance approach eliminates the effectiveness of P-SIFEM. That is, the authors use different numbers of checkpoints during a job's execution to measure the impact on the overhead related to the completion time and deployment cost for different combinations of job length, job memory footprint, and number of revocations. In general, the time/cost overhead mainly falls into four categories: 1) the startup time/cost

overhead that represents additional startup time/cost, which occurs when starting a new spot instance after each revocation; 2) the re-execution time/cost overhead that represents the lost work for each revocation (i.e., lost work using P-SIFEM refers to unsaved and executed work from the beginning of a job, whereas lost work using the fault-tolerance approach refers to unsaved and executed work from the last checkpoint); 3) the checkpointing time/cost overhead that represents the time/cost to checkpoint a job's container into remote storage (i.e., AWS S3); and 4) the recovery time/cost overhead that represents the time/cost to restore a checkpoint of a job's container from remote storage (i.e., AWS S3) into a container deployed on a spot instance for each revocation. Furthermore, the time overhead is divided into the startup time, the job's re-execution time, the job's checkpointing time, and the job's recovery time (i.e., checkpoint restoring time). The cost overhead is divided into the startup cost, the job re-execution cost, the job checkpointing cost, and the job recovery cost (i.e., checkpoint restoring cost). Both the P-SIFEM and the fault-tolerance approach encounter the time/cost of startup overhead and the time/cost of re-execution overhead, whereas the time/cost of checkpointing overhead and the time/cost of recovery overhead are only encountered by the fault-tolerance approach. Understanding whether the optimized number of checkpoints reduces the deployment cost and completion time compared to a random number of checkpoints and P-SIFEM enables us to answer RQ3.

The authors evaluate P-SIFEM in simulations and use Amazon spot instances that contain jobs in Docker containers (Docker Hub, 2022) and realistic price traces from EC2 markets (EC2 markets, 2022). The experiments for the subject applications were performed using spot instances from Amazon EC2 (EC2 Spot Instances, 2022) called r5.2xlarge with an 8 GHz CPU and 64 GB of memory. The authors use Amazon spot instances since their lifetime often exceed hundreds of hours, unlike the lifetime for Google preemptible instances that are less than 24 hours (Sharma et al., 2019). P-SIFEM uses EC2's REST API (Amazon EC2 API, 2022) to collect realistic price traces for all spot instances across all markets (i.e., availability zones and regions) for three months from June 2019 to September 2019. Also, the authors use a load generator called Lookbusy (Carraway, 2022) to create synthetic jobs with different amounts of resource usage. The authors use the memory size to determine suitable sizes of spot instances that are supported by EC2 markets (EC2 markets, 2022). The authors use the memory size to determine suitable spot instances as the memory maintains the state of a running application, which has a significant influence on the overhead related to application completion time (i.e., application's checkpointing time). Then, the authors package jobs in Docker containers that run on Ubuntu 18.04 LTS with a limited CPU and memory capacity for the provisioned spot instances to assess the effectiveness of P-SIFEM for different job memory footprints and job execution lengths. The authors use Docker containers since they support checkpointing and restoring container images and facilitate taking checkpoints at a periodic interval. P-SIFEM proactively checkpoints/restores the image of a job's container deployed on a spot instance to/from an AWS S3 storage (Amazon S3, 2022). In addition, the authors carried out each experiment 100 times and picked the median values for the completion time and the deployment costs for the subject applications, using P-SIFEM and the fault-tolerance approach. All experiments were performed on the same experimental platform to ensure a fair comparison between P-SIFEM and the fault-tolerance approach. The authors used the following checkpointing settings: the number of checkpoints is equal to the number of billing cycles of a job's execution length because the deployment cost relies on the number of billing cycles instead of the actual completion time of the job.

Threats to Validity

Amazon recently changed its pricing policy, such that customers are not required to place bids and that spot prices are based on supply and demand, and as a result, revocations are no longer correlated to customer bids. While this seems a potential threat, it is unlikely a major one since P-SIFEM uses the corresponding on-demand prices instead of customer bids to compute lifetimes. Indeed, users are not willing to pay above the corresponding on-demand price for renting a spot instance to run their applications.

Another potential threat to our empirical evaluation is that our experiments were conducted only on batch job applications, which may make it difficult to generalize the results of the experiments to other types of applications (e.g., interactive job applications) that may have various workflows and behaviors. However, cloud spot instances are often used to run batch job applications. As a result, the authors expect the results of the experiments to be generalizable.

The authors experimented with only a checkpointing-based fault-tolerant approach, whereas other types of fault-tolerant approaches (e.g., live migration) could also result in different effects on the deployment cost and completion time of jobs. However, a live migration requires a limited size of an application's memory footprint (i.e., 4 GB) to complete a migration within the two-minute revocation notice time (Subramanya et al., 2015). P-SIFEM processes jobs in a single spot instance at a time; hence, evaluating P-SIFEM using homogeneous/heterogeneous instances in a cluster for processing jobs is beyond the scope of this work and will be considered in future work.

Another threat to validity is that our experiments were performed in a simulation environment. While this is a potential threat, it is unlikely a major one since the average revocation time of spot instances in a cloud environment (e.g., Amazon EC2) often exceeds hundreds of hours (Sharma et al., 2017), which makes it difficult to assess the effectiveness of P-SIFEM for smaller job execution lengths that often reflect those in production (Google, 2022). That is, the authors use realistic price traces from the Amazon cloud to define the revocation probability of spot instances for all spot instances across all markets (i.e., availability zones and regions) over three months. Additionally, the authors use a realistic time to restore/checkpoint a Docker container deployed on a spot instance in Amazon EC2 to seed our P-SIFEM. For example, the authors measure the time to restore/checkpoint a Docker container that packages jobs with different job execution lengths and job memory footprints in/from S3 storage in Amazon EC2.

Additionally, our experiments were performed only on Docker containers. While this is a potential threat, it is unlikely a major one since P-SIFEM is perfectly applicable to other types of containers, such as Linux Containers, as long as those containers support checkpointing and container image restoration. The authors experimented with only constant jobs, whereas phased jobs in which the utilization of resources varies significantly during various execution phases could also result in different effects on the deployment cost and completion time of jobs. In contrast, understanding the effect of phased jobs on the deployment cost and completion time is beyond the scope of this study and should be considered in future studies.

The authors experimented with a certain price ratio between spot and on-demand instances that is based on realistic price traces from EC2 markets, whereas other ratios between spot and on-demand instances could result in different effects on the deployment cost and completion time of jobs when spot instances are provisioned using P-SIFEM and the fault-tolerance approach. However, understanding the effect of various price ratios between spot instances and on-demand instances is beyond the scope of this study and should be considered in future studies.

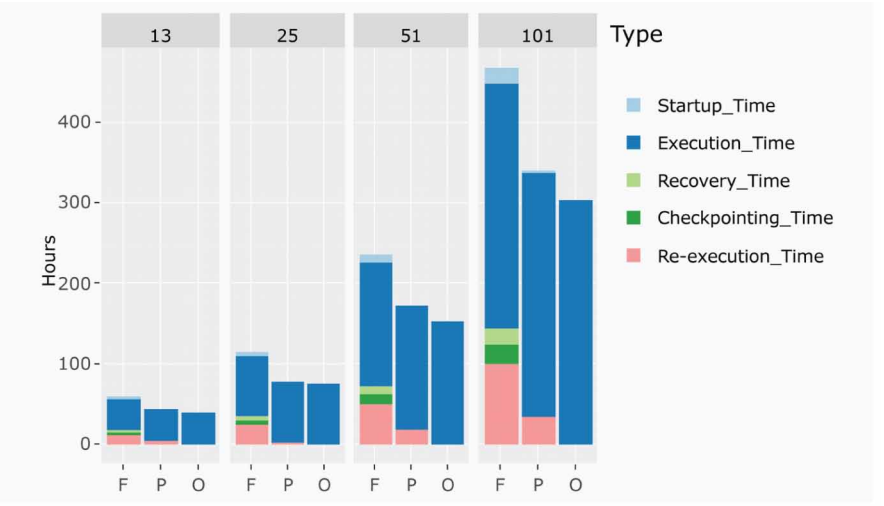
RESULTS

In this section, the authors describe and analyze the results of the experiments to answer the RQs listed in the Evaluation section.

Completion Time

The experimental results that summarize the completion time for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different job execution lengths are shown in the stacked bar plots in Figure 6. The authors observe that the completion time using P-SIFEM is consistently shorter than the completion time using the fault-tolerance approach, and the completion time using P-SIFEM is consistently near that of on-demand instances, which do not incur any additional overhead (Amazon, 2022). This result shows that a longer job length leads

Figure 6. Comparing the completion time for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different job execution lengths while keeping other job features constant



to a steadily higher overhead of completion time resulting from the job's checkpointing, recovery, and re-execution times, as well as the startup time of a spot instance when using the fault-tolerance approach. However, a longer job length leads to a slightly higher overhead of the completion time as a result of the job's re-execution time and the startup time of a spot instance when using P-SIFEM. Our explanation is that P-SIFEM does not incur frequent job re-execution time, and the startup time of a spot instance using P-SIFEM does not increase with the increase in job execution length. This is expected based on the way P-SIFEM provisions a spot instance with a high lifetime.

The experimental results that summarize the completion time for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different job memory footprints are shown in the stacked bar plots in Figure 7. The authors observe that the completion time for P-SIFEM is consistently shorter than the completion time for the fault-tolerance approach, and the completion time for P-SIFEM is consistently near that of on-demand instances, which do not incur any additional overhead (Amazon, 2022). This result shows that a larger job memory footprint leads to a higher overhead of the completion time resulting from the job's checkpointing time and recovery time when using the fault-tolerance approach. In contrast, the overhead of the completion time resulting from the job's re-execution time and the startup time of a spot instance when using the fault-tolerance approach stays approximately the same across various job memory footprints, which suggests that the overhead resulting from the job's re-execution time and the startup time of a spot instance for the fault-tolerance approach is independent of the job resource usage. Additionally, the overhead of an application's completion time resulting from the job's re-execution time and the startup time of a spot instance when using P-SIFEM stays approximately the same across various job memory footprints, which suggests that the completion time for the subject applications when using P-SIFEM is also independent of the resource usage.

The experimental results that summarize the completion time for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different numbers of revocations are shown in the stacked bar plots in Figure 8. The authors observe that the completion time for P-SIFEM—except for when the number of revocations equals one—is consistently shorter than the completion time for the fault-tolerance approach, and the completion time for P-SIFEM is consistently near that of on-demand instances, which do not incur any additional overhead (Amazon, 2022). When the number of revocations equals one, the job's checkpointing time for the fault-tolerance

Figure 7. Comparing the completion time for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different memory footprints while keeping other job features constant

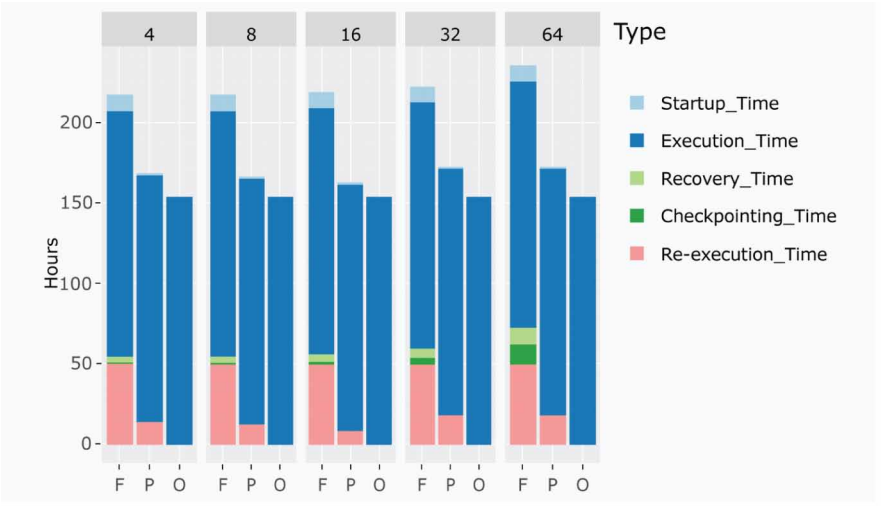
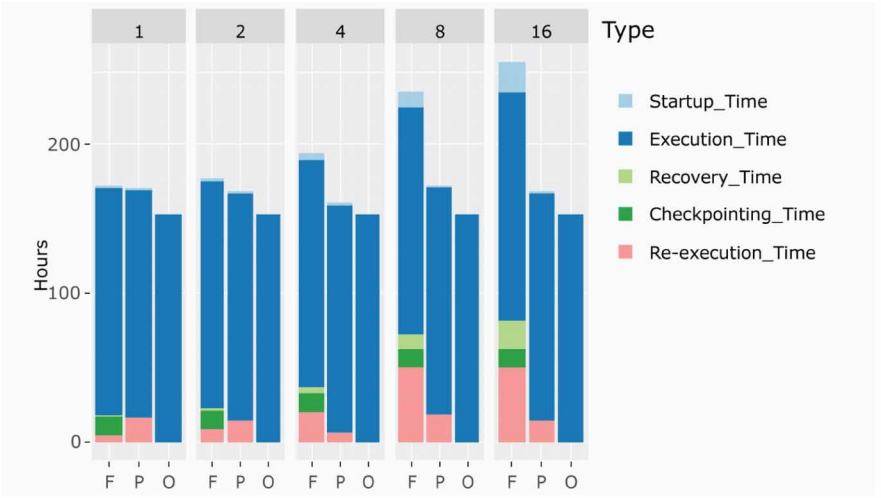


Figure 8. Comparing the completion time for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different revocation numbers while keeping other job features constant



approach balances the job's re-execution for P-SIFEM. This result suggests that the fault-tolerance approach incurs additional overhead due not only to the number of revocations but also to the number of checkpoints. It also suggests that the effectiveness of P-SIFEM may decrease when the number of revocations decreases, and it is very difficult to guarantee that the number of revocations is small (Shastri et al., 2017). The job's recovery time, the job's re-execution time, and the startup time of a spot instance—except for the job's checkpointing time—all increase steadily when using the fault-tolerance approach, whereas in P-SIFEM, the job's re-execution time and the startup time of a spot instance remain approximately the same. This observation suggests that the job's checkpointing time for the fault-tolerance approach as well as the job's re-execution time and the startup time of a spot instance for P-SIFEM are independent of the number of revocations. In summary, these experimental

results allow us to conclude that P-SIFEM is more efficient in executing applications for different job execution lengths, job memory footprints, and numbers of revocations than the fault-tolerance approach, thus positively addressing RQ1.

Deployment Costs

The experimental results that summarize the deployment costs for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different job execution lengths are shown in the stacked bar plots in Figure 9. The authors observe that the deployment costs using P-SIFEM are consistently lower than the deployment costs using the fault-tolerance approach or those of on-demand instances. This result identifies the steady rise in overhead related to deployment costs that result from the job's checkpointing costs, its recovery costs, its re-execution costs, the startup costs of spot instances, and the buffer costs of billing cycles when using the fault-tolerance approach with the increased job length. However, this result also identifies a slight rise in the overhead of deployment costs that result from the job's re-execution cost, the startup costs of spot instances, and the buffer costs of billing cycles when using P-SIFEM with the increased length. Our explanation is that P-SIFEM does not frequently incur the job's re-execution costs and the startup costs of spot instances since the startup costs of spot instances using P-SIFEM do not increase with the increase of the job execution length, which is expected based on the way that P-SIFEM provisions a spot instance with a high lifetime. Interestingly, the authors observe that unlike P-SIFEM, the buffer costs of billing cycles significantly increase compared to the other types of overhead costs when using the fault-tolerance approach with the increase of the job length, which suggests that the fault-tolerance approach incurs not only overhead related to the settings of the fault tolerance approach (e.g., the job's checkpointing cost) but also additional overhead related to the cloud billing policies (i.e., the buffer costs of billing cycles). Additionally, the authors observe that the deployment costs of the fault-tolerance approach across all job lengths are equal to or higher than the deployment costs of on-demand instances (Amazon, 2022), which suggests that using on-demand for larger job lengths may reduce deployment costs and the completion time when compared to the fault-tolerance approach.

The experimental results that summarize the deployment costs for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different job memory footprints are shown in the stacked bar plots in Figure 10. The authors observe that the deployment costs using

Figure 9. Comparing the deployment costs for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different job execution lengths while keeping other job features constant

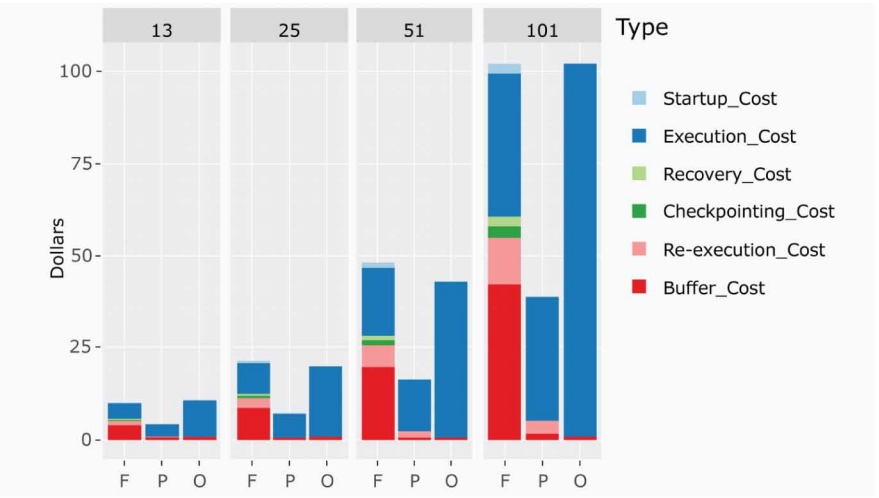
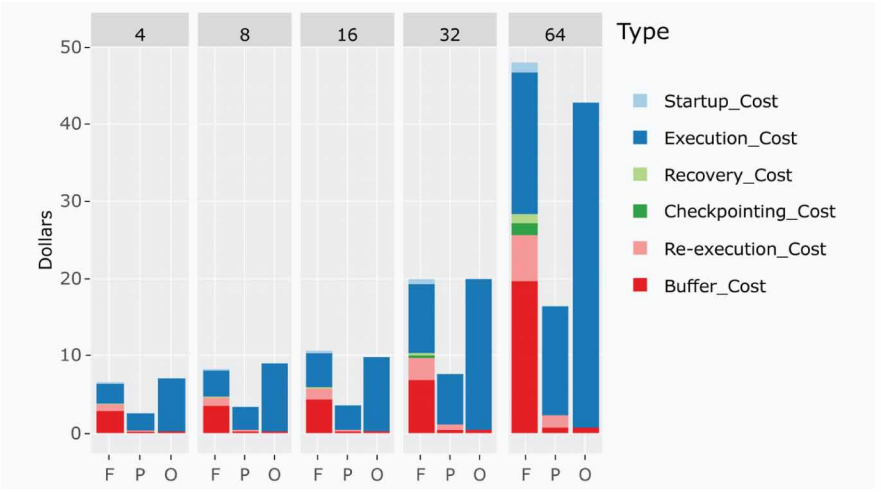


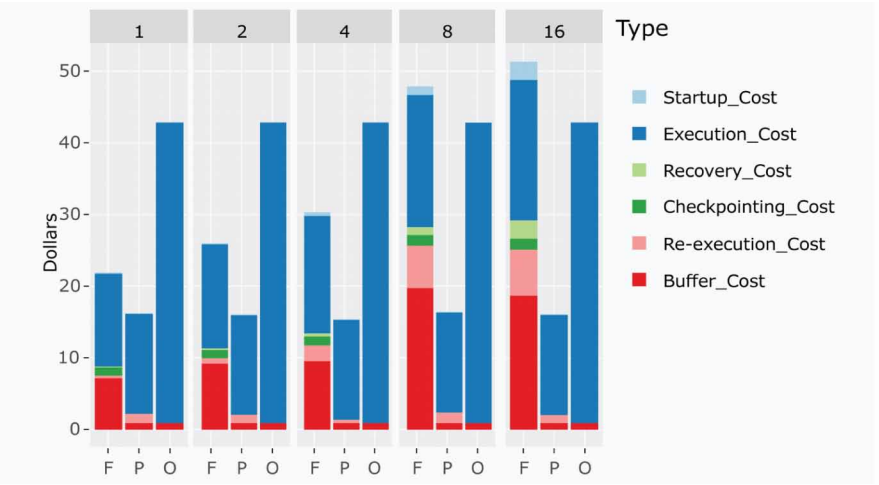
Figure 10. Comparing the deployment costs for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different memory footprints while keeping other job features constant



P-SIFEM are consistently lower than the deployment costs using the fault-tolerance approach and on-demand instances. This result demonstrates the steady rise of the overhead related to deployment costs resulting from the job's checkpointing, recovery, re-execution, and startup costs of spot instances, as well as the buffer costs of billing cycles when using the fault-tolerance approach with the increase of job memory footprint. However, this result demonstrates a slight rise in the overhead of deployment costs resulting from the job's re-execution and startup costs of spot instances and the buffer costs of billing cycles when using P-SIFEM with the increase in the job memory footprint. Our explanation is that P-SIFEM does not incur the job's re-execution and startup costs of spot instances, since the startup costs of spot instances using P-SIFEM do not increase with the increase of the job memory footprint, which is expected based on the way that P-SIFEM provisions a spot instance with a high lifetime. The authors observe that, unlike the buffer costs of billing cycles for P-SIFEM, the buffer costs of billing cycles for the fault-tolerance approach significantly increase with the higher job memory footprints (i.e., 32 and 64 GB), suggesting that the buffer costs increase when there is a significant change in deployment time between consecutive job memory footprints (i.e., exceeds the period for a billing cycle). Additionally, the authors observe that the deployment costs of the fault-tolerance approach across all job memory footprints are equal to or higher than the deployment costs of on-demand instances (Amazon, 2022), which suggests that provisioning on-demand for large job memory footprints may result in lower deployment costs and completion time than the fault-tolerance approach.

The experimental results that summarize the deployment costs for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different numbers of revocations are shown in the stacked bar plots in Figure 11. The authors observe that the deployment costs using P-SIFEM and that of on-demand instances are consistently lower than the deployment costs using the fault-tolerance approach. The job's recovery and re-execution costs, the startup costs of spot instances, and the buffer costs of billing cycles, except for the job's checkpointing costs, increase steadily when using the fault-tolerance approach, whereas for P-SIFEM, the job's re-execution costs, the startup costs of spot instances, and the buffer costs of billing cycles remain approximately the same. This observation suggests that the job's recovery time and re-execution costs, the startup costs of spot instances, and the buffer costs of billing cycles depend on the number of revocations when using the fault-tolerance approach. However, the job's checkpointing costs for the fault-tolerance approach and the job's re-execution costs, the startup costs of spot instances, and the buffer costs of

Figure 11. Comparing the deployment costs for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different revocation numbers while keeping other job features constant



billing cycles for P-SIFEM are independent of the number of revocations. Our explanation is that P-SIFEM does not incur the job's re-execution costs and the startup costs of spot instances. The authors observe that unlike the buffer costs of billing cycles for P-SIFEM, the buffer costs of billing cycles for the fault-tolerance approach significantly increase with the higher numbers of revocations (i.e., 8 and 16), which suggests that the buffer costs increase when there is a significant change in deployment time between consecutive numbers of revocations (i.e., exceeds the period for a billing cycle). Interestingly, the authors observe that the deployment costs for the fault-tolerance approach when the number of revocations is high (i.e., 8 and 16) are significantly higher than the deployment costs for on-demand instances (Amazon, 2022), which confirms that provisioning on-demand for a large number of revocations may result in lower deployment costs and completion time than the fault-tolerance approach. In summary, these experimental results allow us to conclude that P-SIFEM is more effective in reducing the deployment costs of applications for different job execution lengths, job memory footprints, and numbers of revocations than the fault-tolerance approach, thus positively addressing RQ2.

Impact on Different Numbers of Checkpoints

The experimental results that summarize the completion time and deployment cost for the subject applications using P-SIFEM, the fault-tolerance approach, and on-demand instances for different numbers of checkpoints are shown in the stacked bar plots in Figure 12 and Figure 13, respectively. The authors observe that the completion time and deployment cost using P-SIFEM are consistently lower than the completion time and deployment cost using the fault-tolerance approach. The completion time using P-SIFEM is consistently near that of the on-demand instance, and the deployment cost using P-SIFEM is consistently lower than the deployment cost using on-demand instances. This result shows that a higher number of checkpoints leads to a steadily higher overhead related to application completion time and deployment cost resulting from the job's checkpointing time and cost, which is expected because the job's checkpointing time and cost depend on the number of checkpoints. Interestingly, the authors observe that the completion time and deployment cost for the lowest number of checkpoints is almost equal to the completion time and deployment cost for the highest number of checkpoints. Our explanation is that the job's re-execution time and cost for the lowest number of checkpoints balance the job's checkpointing time and cost for the highest number of checkpoints.

Figure 12. Comparing the completion time for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different numbers of checkpoints while keeping other job features constant

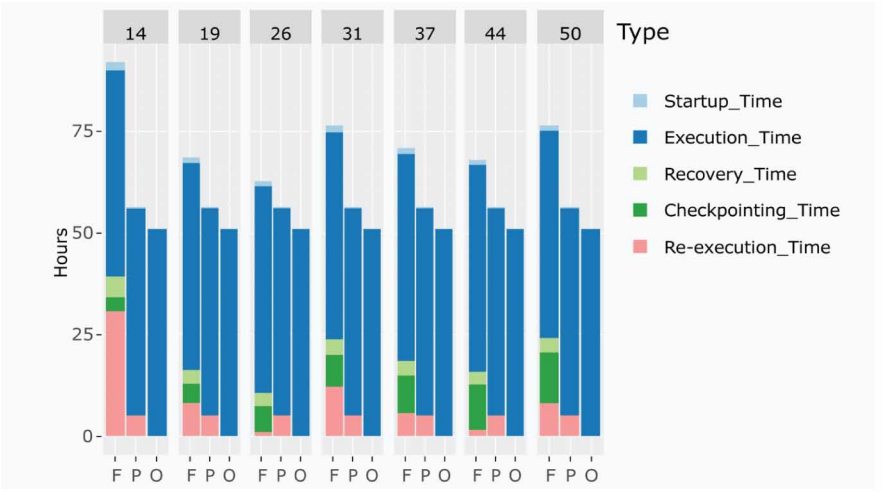
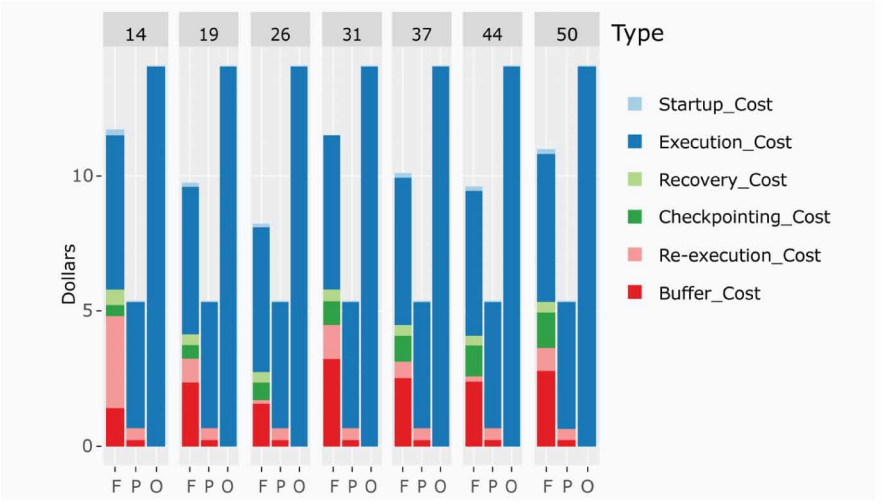


Figure 13. Comparing the deployment costs for the subject applications using P-SIFEM (P), the fault-tolerance approach (F), and on-demand instances (O) for different numbers of checkpoints while keeping other job features constant



This result suggests that the fault-tolerance approach incurs additional overhead due to not only a high number of checkpoints but also a low number of checkpoints. The authors observe the lowest completion time and deployment cost when the number of checkpoints is 26, which suggests that the optimal tradeoff between the overhead of actual checkpointing and the overhead of re-executing the lost work from the last checkpoint on each spot instance revocation when the number of checkpoints is 26. Although this optimal number of checkpoints reduces the completion time and deployment cost compared to the other number of checkpoints, this optimal number of checkpoints (i.e., the settings of the fault-tolerance approach) does not reduce the completion time and deployment cost compared to P-SIFEM. This result confirms that optimizing the settings of a fault-tolerance approach does not eliminate the effectiveness of P-SIFEM, thus positively addressing RQ3.

Discussion

The authors address a challenging problem for applications deployed on cloud spot instances that results from the overhead of employing fault-tolerance mechanisms. The authors propose a novel cloud market-based approach for Provisioning Spot Instances using FEatures of cloud Markets (P-SIFEM) to reduce the deployment cost and completion time of applications. To assess the benefit of our approach, the authors evaluate P-SIFEM with different combinations of job settings (i.e., job execution length and job memory footprint), thus enabling us to answer RQ1 and RQ2. Our results in the completion time section allow us to conclude that P-SIFEM is more efficient in executing applications for different job execution lengths, job memory footprints, and numbers of revocations than the fault-tolerance approach, thus positively addressing RQ1. Our results in the deployment cost section allow us to conclude that P-SIFEM is more effective in reducing the deployment costs of applications for different job execution lengths, job memory footprints, and numbers of revocations than the fault-tolerance approach, thus positively addressing RQ2. Also, the authors determine whether optimizing the number of checkpoints reduces the deployment cost and completion time compared to a random number of checkpoints, thus enabling us to answer RQ3. Our results in the section of impact on different numbers of checkpoints confirm that optimizing the settings of a fault-tolerance approach does not eliminate the effectiveness of P-SIFEM, thus positively addressing RQ3. In general, our results show that P-SIFEM reduces the deployment cost and completion time compared to approaches based on fault-tolerance mechanisms. As a result, our results confirm that our hypothesis of leveraging cloud market features without employing fault-tolerance mechanisms to provision spot instances for applications reduces the deployment cost compared to the deployment cost using fault-tolerance approaches or on-demand instances and maintains the completion time near that of on-demand instances.

In the future, the authors plan to build an availability-driven model to control the lack of resources for applications deployed on cloud spot instances based on noncooperative game theoretic approaches (e.g., Nash equilibrium game theory), which incorporate spot price traces (e.g., historical bids) and properties of users and applications. The purpose is to predict user demand best in terms of both spot price and application execution to maximize the revenue for cloud providers and users. The authors also plan to study the relationships/dependencies between these properties and their impacts on the spot price to build a bidding strategy model that optimizes the utilization of spot instances. The authors could also take advantage of the Monte Carlo simulation, state representation methodology, and VM allocation scheme to build an optimal bidding model.

CONCLUSION

The authors addressed a challenging problem for applications deployed on cloud spot instances that results from the overhead of employing fault-tolerance mechanisms. The authors proposed a novel cloud market-based approach for Provisioning Spot Instances using FEatures of cloud Markets (P-SIFEM) to reduce the deployment cost and completion time of applications. The authors evaluated P-SIFEM in simulations and used Amazon spot instances that contain jobs in Docker containers and realistic price traces from EC2 markets. Our simulation results show that our approach reduces the deployment cost and completion time compared to approaches based on fault-tolerance mechanisms.

ACKNOWLEDGMENT

We warmly thank the editor, Prof. Brij Gupta and anonymous reviewers for their comments and suggestions that helped us to improve the quality of this paper.

CONFLICT OF INTEREST

The authors of this publication declare there is no conflict of interest.

FUNDING AGENCY

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

Ahammad, I., Khan, M. A. R., Salehin, Z. U., Uddin, M., & Soheli, S. J. (2021). Improvement of QOS in an IoT ecosystem by integrating fog computing and SDN. *International Journal of Cloud Applications and Computing*, *11*(2), 48–66.

Ahuja, S. P., Czarnecki, E., & Willison, S. (2020). Multi-factor performance comparison of amazon web services elastic compute cluster and google cloud platform compute engine. *International Journal of Cloud Applications and Computing*, *10*(3), 1–16.

Aliyu, M., Murali, M., Gital, A. Y., & Boukari, S. (2020). Efficient metaheuristic population-based and deterministic algorithm for resource provisioning using ant colony optimization and spanning tree. *International Journal of Cloud Applications and Computing*, *10*(2), 1–21.

Alourani, A., Kshemkalyani, A. D., & Grechanik, M. (2020). T-basir: Finding shutdown bugs for cloud-based applications in cloud spot markets. *IEEE Transactions on Parallel and Distributed Systems*, *31*(8), 1912–1924. doi:10.1109/TPDS.2020.2980265

Amazon. (2022, June). *Amazon ec2 instances*. https://aws.amazon.com/ec2/

Amazon S3. (2022, June). *Amazon simple storage service*. https://aws.amazon.com/s3

Amazon EC2 API. (2022, June). *Amazon EC2 API Reference*. https://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html

Bisht, J., & Vampugani, V. S. (2022). Load and Cost-Aware Min-Min Workflow Scheduling Algorithm for Heterogeneous Resources in Fog, Cloud, and Edge Scenarios. *International Journal of Cloud Applications and Computing*, *12*(1), 1–20.

Carraway, D. (2022, June). *Lookbusy – A synthetic load generator*. http://www.devin.com/lookbusy

Dharwadkar, N. V., Poojara, S. R., & Kadam, P. M. (2018). Fault tolerant and optimal task clustering for scientific workflow in cloud. *International Journal of Cloud Applications and Computing*, *8*(3), 1–19.

Docker Hub. (2022, June). *Docker Container Images*. https://hub.docker.com/u/library/

EC2 Markets. (2022, June). *AWS EC2 Spot Instance Price Histories*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances-history.html

EC2 Spot Instances. (2022, June). *Amazon EC2 Spot Instances*. https://aws.amazon.com/ec2/spot

Funaro, L., Ben-Yehuda, O. A., & Schuster, A. (2019, April). Stochastic resource allocation. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (pp. 122-136). ACM.

Gond, S., & Singh, S. (2019). Dynamic load balancing using hybrid approach. *International Journal of Cloud Applications and Computing*, *9*(3), 75–88.

Google. (2022, June). *Cluster workload traces*. https://github.com/google/cluster-data

Goundar, S., & Bhardwaj, A. (2018). Efficient fault tolerance on cloud environments. *International Journal of Cloud Applications and Computing*, *8*(3), 20–31. doi:10.4018/IJCAC.2018070102

Harrath, Y., & Bahlool, R. (2019). Multi-objective genetic algorithm for tasks allocation in cloud computing. *International Journal of Cloud Applications and Computing*, *9*(3), 37–57.

Herzfeldt, A. B., Rauer, H. P., Weißbach, R., & Ertl, C. (2020). Cloud Computing as the Next Utility: Market Strategies for Cloud Service Providers. *International Journal of Cloud Applications and Computing*, *10*(4), 28–47. doi:10.4018/IJCAC.2020100103

Jaswal, S., & Malhotra, M. (2022). AFTTM: Agent-Based Fault Tolerance Trust Mechanism in Cloud Environment. *International Journal of Cloud Applications and Computing*, *12*(1), 1–12. doi:10.4018/IJCAC.297105

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.

Kapgate, D. (2021). Predictive data center selection scheme for response time optimization in cloud computing. *International Journal of Cloud Applications and Computing*, *11*(1), 93–111.

Khodak, M., Zheng, L., Lan, A. S., Joe-Wong, C., & Chiang, M. (2018, April). Learning cloud dynamics to optimize spot instance bidding strategies. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (pp. 2762-2770). IEEE.

Mishra, A. K., Kesarwani, A., & Yadav, D. K. (2019, 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.

Nandal, P., Bura, D., Singh, M., & Kumar, S. (2021). Analysis of Different Load Balancing Algorithms in Cloud Computing. *International Journal of Cloud Applications and Computing*, *11*(4), 100–112.

New Spot Instance Pricing. (2022, June). *New Amazon Web Services EC2 Spot Instance Pricing*. https://aws.amazon.com/about-aws/whats-new/2017/11/\\amazon-ec2-spot-introduces-new-pricing-model-and-the-ability-to-launch-new-spot-instances-via-runinstances-api/

Panda, S. K., & Naik, S. (2018). An efficient data replication algorithm for distributed systems. *International Journal of Cloud Applications and Computing*, *8*(3), 60–77. doi:10.4018/IJCAC.2018070105

Priyanka, H., & Cherian, M. (2021). Effective Utilization of Resources Through Optimal Allocation and Opportunistic Migration of Virtual Machines in Cloud Environment. *International Journal of Cloud Applications and Computing*, *11*(3), 72–91. doi:10.4018/IJCAC.2021070105

Sahana, S., Mukherjee, T., & Sarddar, D. (2020). A conceptual framework towards implementing a cloud-based dynamic load balancer using a weighted round-robin algorithm. *International Journal of Cloud Applications and Computing*, *10*(2), 22–35.

Sharma, P., Ali-Eldin, A., & Shenoy, P. (2019, March). Resource deflation: A new approach for transient resource reclamation. In *Proceedings of the Fourteenth EuroSys Conference 2019* (pp. 1-17). Academic Press.

Sharma, P., Irwin, D., & Shenoy, P. (2017). Portfolio-driven resource management for transient cloud servers. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *1*(1), 1-23.

Shastri, S., & Irwin, D. (2017, September). Hotspot: automated server hopping in cloud spot markets. In *Proceedings of the 2017 Symposium on Cloud Computing* (pp. 493-505). Academic Press.

Singh, G., Malhotra, M., & Sharma, A. (2022). An adaptive mechanism for virtual machine migration in the cloud environment. *International Journal of Cloud Applications and Computing*, *12*(1), 1–10. doi:10.4018/IJCAC.297095

Soltani, B., Ghenai, A., & Zeghib, N. (2020). Execution of long-duration multi-cloud serverless functions using selective migration-based approach. *International Journal of Cloud Applications and Computing*, *10*(4), 70–97.

Subramanya, S., Guo, T., Sharma, P., Irwin, D., & Shenoy, P. (2015, August). Spoton: a batch computing service for the spot market. In *Proceedings of the sixth ACM symposium on cloud computing* (pp. 329-341). ACM.

Swarnakar, S., Bhattacharya, S., & Banerjee, C. (2021). A bio-inspired and heuristic-based hybrid algorithm for effective performance with load balancing in cloud environment. *International Journal of Cloud Applications and Computing*, *11*(4), 59–79.

Vinothina, V., & Rajagopal, S. (2022). Review on Mapping of Tasks to Resources in Cloud Computing. *International Journal of Cloud Applications and Computing*, *12*(1), 1–17. doi:10.4018/IJCAC.2022010106

Youssef, F., El Habib, B. L., Hamza, R., El Houssine, L., Ahmed, E., & Hanoune, M. (2018). A new conception of load balancing in cloud computing using tasks classification levels. *International Journal of Cloud Applications and Computing*, *8*(4), 118–133.

Abdullah Alourani is an assistant professor at the Department of Computer Science and Information, Majmaah University, Saudi Arabia. He received his Ph.D. in computer science from the University of Illinois at Chicago, United States, his Master's degree in computer science from DePaul University in Chicago, United States, and his Bachelor's degree in computer science from Qassim University, Saudi Arabia. His current research interests are in the areas of cloud computing, distributed systems, and software engineering. He is a member of ACM and IEEE.

Ajay D. Kshemkalyani received the BTech degree in computer science from the Indian Institute of Technology, Bombay, in 1987, and the PhD degree in computer science from the Ohio State University, in 1991, respectively. He is currently a professor with the Department of Computer Science, University of Illinois at Chicago. His research interests include distributed computing, computer networks, and concurrent systems. In 1999, he received the US National Science Foundation Career Award. He has served on the editorial board of the Elsevier journal Computer Networks, and the IEEE Transactions on Parallel and Distributed Systems. He has coauthored a book entitled Distributed Computing: Principles, Algorithms, and Systems (Cambridge University Press, 2011). He is a distinguished scientist of the ACM and a senior member of the IEEE.