Towards Online Checkpointing Mechanism for Cloud Transient Servers

Luping Wang, Wei Wang, Bo Li
Hong Kong University of Science and Technology
{lwangbm, weiwa, bli}@cse.ust.hk

***Abstract*—Cloud providers such as Amazon EC2 and Google GCE typically price their idle compute resources at a significant discount in the form of *transient servers*. Unlike regular servers, cloud transient servers have no reliability guarantee and can be revoked *anytime*. In order to enjoy the price premium of transient servers while mitigating the impact of indefinite server revocations, cloud users continuously checkpoint current computational states to stable storage. To determine the optimal checkpointing interval, prior works have largely built upon predicting future revocations. However, we argue that existing prediction models may not work well if providers employ different server revocation algorithms in the future, and the required historical information can be unavailable in practice. In this work, we propose two *online* algorithms, one *competitive* and the other *heuristic*, that determine the checkpointing intervals without predicting future revocation based on history. Our competitive online algorithm achieves the *best possible* competitive ratio against the optimal, yet impractical, offline solution. Our heuristic algorithm adaptively reacts to server revocations, achieving better average performance than the competitive algorithm, especially for short-running tasks. Extensive trace-driven simulations confirm our analytical results and demonstrate the efficacy of the two online algorithms in the context of Amazon EC2 Spot Instances.**

I. Introduction

Provisioning idle servers in datacenters can be expensive to Infrastructure-as-a-Service (IaaS) cloud providers with non-trivial operating cost. To minimize the revenue loss, IaaS providers, such as Amazon and Google, price their idle compute resources with a significant discount in the form of *transient servers*, e.g., Amazon EC2 Spot Instances [1] and Google GCE Preemptible Instances [2]. Unlike regular instances, transient servers provide no reliability guarantee and are subject to *indefinite revocations*. For example, Amazon EC2 leases spare capacity in an auction-like market and charges users a dynamic *spot price*: Whenever the spot price rises above a user's bid, the Spot instances of that user get revoked [1].

In order to take advantage of the significant price premium of transient servers, cloud users employ *checkpoint-restart* mechanisms to minimize the impact of indefinite instance revocations. By continuously checkpointing computational states to stable storage (e.g., Amazon S3), a failed task can quickly resume from the recent checkpoint after being rescheduled to another server, thus reducing the recomputational overhead caused by revocations.

However, there is a fundamental trade-off between recomputational cost and checkpointing overhead. Sheng et al. [3] showed that the time to checkpoint an instance's computational state is linearly dependent to its memory footprint, ranging from several seconds to tens of minutes. The more frequently the checkpointing is made, the more time is spent on data persisting, yet the faster a failed task can get recovered from server revocation. Therefore, a checkpointing algorithm is needed for cloud transient servers that determines, dynamically, when the checkpointing should be made such that the recomputational overhead is minimized without wasting too much time persisting data to stable storage.

Existing checkpointing mechanisms for transient servers are largely built on predicting the next revocation occurrence based on historical information [3]–[8]. We argue that these prediction-based approaches suffer from the following two major limitations in practice:

1) *Historical information is loosely correlated to future revocations.* In order to model the revocation of Amazon EC2 Spot Instances, many works focus on predicting the spot price based on history [4]–[7]. However, it has been suggested that the EC2 Spot prices exhibit an intrinsic randomness engineered by some internal dynamic algorithms [9]. Even if these algorithms can be fully characterized by existing prediction models, future algorithm updates would likely invalidate all of them.
2) *Historical information on server revocations may not be available.* Unlike Amazon EC2, the transient servers in Google GCE, called Preemptible Instances, are subject to a fixed discount yet can be revoked anytime. However, no revocation history is made available by Google.

In light of the two limitations of existing works, we ask a question: Is it possible to design an *online* checkpointing algoirthm that does not assume any prediction models of future revocation, while striking a good balance between checkpointing overhead and recomputational cost?

We answer this question with two online checkpointing algorithms, one *competitive* and the other *heuristic*. We show, analytically, that our competitive algorithm provides *guaranteed performance*: By any time, the computation progress of the proposed algorithm is *at least* $\frac{1}{2}$ of that of the optimal, yet impractical, offline solution assuming the exact knowledge of future revocations. In addition, we prove that our proposal is the *only* online algorithm with performance guarantees, hence achieving the *optimal* competitive ratio $\frac{1}{2}$.

Our second proposal is a novel online *heuristic* algorithm that outperforms the competitive alternative with lower checkpointing overhead *on average* by adaptively reacting to server revocations.

978-1-5090-5019-2/17/$31.00 ©2017 IEEE

We have evaluated the two online algorithms using trace-driven simulations against Amazon EC2 Spot Price traces. Simulation results confirm the $\frac{1}{2}$-competitiveness of our competitive online algorithm. Further studies show that, for workloads with short-running tasks, the average performance can be significantly improved by the heuristic algorithm.

II. Background And Related Work

In this section, we provide background information and briefly survey the related work.

A. Transient Servers in IaaS Cloud

Leading cloud providers, such as Amazon EC2 and Google GCE, lease their idle compute resources as transient servers at a significant discount.

EC2 Spot Instance: In Amazon EC2, transient servers are offered as *Spot instances* in an auction-like market where users bid for spare resources [1]. Based on the demand and supply, EC2 dynamically posts a Spot price to clear the market. Users with higher bids win the auction and are entitled access to Spot instances. However, Spot instances provide no reliability guarantee and are revoked whenever the user becomes *out-of-bid*, i.e., the Spot price rises above its bid.

GCE Preemptible Instance: In Google GCE, transient servers are offered as *preemptible instances* [2]. Unlike EC2 Spot instances, preemptible instances are subject to a *fixed* discount and can be revoked indefinitely. The revocation history of preemptible instances is not available to GCE users.

B. Existing Checkpointing Schemes for Transient Servers

The significant price discount of transient servers and their frequent, indefinite revocations force cloud users to continuously checkpoint computational states to stable storages. Notably, Yi et al. [4] proposed two simple heuristics to determine when the checkpointing should be made: the first checkpoints data in an hourly basis; the second makes checkpointing based on the distribution of past revocations. Jung et al. [5] proposed another simple heuristic for EC2 Spot instances that checkpoints at the rising edge of Spot price. Di et al. [3] studied an *equidistant* checkpointing strategy and derived the optimal checkpointing period based on the statistical mean time between revocations (MTBF). More recently, Gong et al. [6] suggest that complementing checkpointing mechanism with replication techniques can achieve fault tolerance with more salient cost savings for transient servers, if the function of instance revocation rate is available *a priori*. There are also some works [7], [8] that model the dynamics of Spot prices as a mixture of Gaussian distribution and Markov chain. The model can then be used to predict the timing of next revocation.

We note that all these checkpointing strategies are based on predicting future revocations. Their performance critically depend on the accuracy of modeling and the availability of instance revocation history. Unfortunately, as we have explained in the previous section, none of these requirements can be easily met in practice.

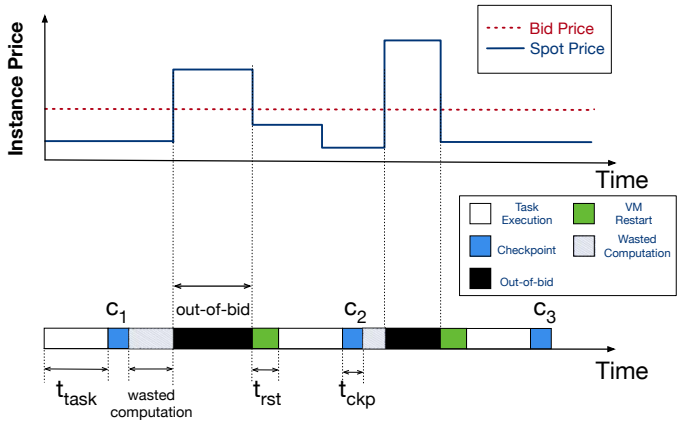


Fig. 1: An example of the discrete checkpointing model for an Amazon EC2 Spot instance.

III. Problem Formulation

In this section, we formulate the checkpointing problem and settle our objective to maximizing the *effective computation time*. We present the optimal, yet impractical, offline solution as a benchmark to measure the performance of other algorithms.

A. Discrete Checkpointing Model

Cloud applications typically run as many parallel tasks on different servers, and the computational states change upon a task completion. We therefore assume a *discrete* checkpointing model: A checkpoint cannot be made anytime, but only after a task completion. Given that tasks of a cloud application are usually copies of the same binary program running on different data partitions (e.g., Hadoop MapReduce), we assume that tasks have the same runtime t_{task}, and it takes time t_{ckp} to checkpoint the output of a task to a stable storage. We further assume that checkpointing is faster than recomputation:

$$t_{\text{ckp}} \leq t_{\text{task}}. \tag{1}$$

Otherwise, users would simply turn to recomputation to achieve fault tolerance. Finally, upon a server failure due to instance revocation, it takes time t_{rst} to reschedule the failed task to another server and restart it from the recent checkpoint.

Fig. 1 illustrates an example of the discrete checkpointing model for an Amazon EC2 Spot instance. We see from the top figure that the dynamically changing Spot prices result in frequent instance revocation due to out-of-bid. The bottom figure shows how checkpointing can be helpful in this example.

B. Problem Formulation

We are now ready to formulate the checkpointing problem. Formally, given a task runtime t_{task}, a checkpointing cost t_{ckp}, a restarting time t_{rst}, and a sequence of failure occurrences due to revocation $F = \{f_1, f_2, \ldots, f_n\}$, a checkpointing algorithm should schedule a sequence of checkpointing $C = \{c_1, c_2, \ldots, c_k\}$. Fig. 2 shows such a discrete checkpointing process in the example of Fig. 1.

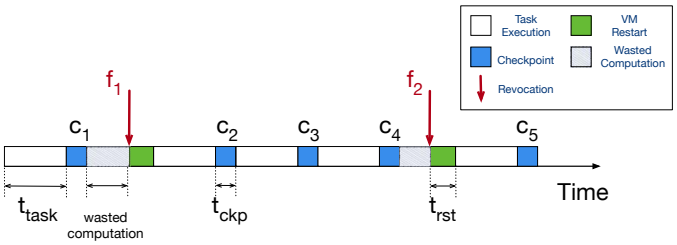


Fig. 2: Discrete checkpointing decision process.

TABLE I: Summary of important notations.

Notation	Description
t_{task}	single task duration in the workload
t_{ckp}	single checkpoint time overhead
t_{rst}	VM restarting time overhead
f_j	the j^{th} revocation occurrence time
c_i	starting time of the i^{th} checkpoint
T_{entire}	entire time duration
T_{EC}	effective computation time
T_i	effective computation the i^{th} checkpoint records
$t_{\text{waste }j}$	lost computation due to the j_{th} revocation

Notations: Let c_i be the time when the i^{th} checkpointing starts, and Δ_i the set of revocations occurred during the i^{th} checkpointing, i.e.,

$$\Delta_i = \{f_j | c_i \leq f_j \leq c_i + t_{\text{ckp}}\}, \tag{2}$$

where f_j denotes the arrival time of the j^{th} revocation. Let c_i' be the time when the i^{th} checkpointing terminates. Note that the checkpointing may be interrupted by an instance revocation. In this case, the checkpointing termination time c_i' is simply the revocation arrival, i.e.,

$$c_i' = \begin{cases} c_i + t_{\text{ckp}} & \Delta_i = \emptyset, \\ \max_{f_j \in \Delta_i} f_j & \Delta_i \neq \emptyset, \end{cases} \tag{3}$$

Let σ_i be the set of revocations between c_{i-1}' and c_i, i.e.,

$$\sigma_i = \{f_j | c_{i-1}' < f_j < c_i\}. \tag{4}$$

Whenever such a revocation occurs, a failed task must be recomputed from the previous checkpoint. Let s_i be the *earliest* time since the previous checkpointing from which there is no revocation until the i^{th} checkpointing, i.e.,

$$s_i = \max\{c_{i-1}', \max_{f_j \in \sigma_i} f_j\}. \tag{5}$$

In our discrete checkpointing decision model, a well justified algorithm should not checkpoint in the middle of task execution, but always after a task completion. Without loss of generality, we impose the following constraints:

$$c_i - s_i = M t_{\text{task}}, \tag{6}$$

where M is an integer.

Table I summarizes some important notations.

Objective: Given the revocation arrivals $F = \{f_1, f_2, \ldots, f_n\}$, our objective is to persist the output of as many tasks as possible. In particular, let T_i be the *effective computation time* retained by the i^{th} checkpointing, which is measured as the compute work persisted since the previous checkpoint. We differentiate between the two cases. If the i^{th} checkpointing has completed successfully without interruption, all the works computed since the last checkpoint are persisted, i.e., $T_i = c_i - s_i$. Otherwise, we lose all works, and $T_i = 0$. To summarize, we have

$$T_i = \begin{cases} c_i - s_i, & \Delta_i = \emptyset; \\ 0, & \Delta_i \neq \emptyset. \end{cases} \tag{7}$$

The total *effective computation time* is simply the work persisted by all checkpoints, i.e.,

$$T_{\text{EC}} = \sum_i T_i. \tag{8}$$

Our objective is to design an optimal checkpointing algorithm that maximizes the effective computation time, i.e.,

$$\text{maximize} \quad T_{\text{EC}}. \tag{9}$$

We emphasize that in practice, the checkpointing decisions must be made *online*, without any knowledge of future revocations.

C. Optimal Offline Solution

In the online algorithm literature [10], the optimal, yet impractical, *offline* solution is used as a benchmarking algorithm to measure the performance of different online solutions. Back to our problem, an offline solution assumes the entire knowledge of instance revocation sequence $F = \{f_1, f_2, \ldots, f_n\}$. With this information, the optimal checkpointing decisions can be trivially obtained. The algorithm simply checkpoints before the next revocation occurrence. In particular, we compute the maximum number of tasks that can complete between the two consecutive revocations f_j and f_{j+1}:

$$l = \lfloor (f_{j+1} - f_j - t_{\text{ckp}} - t_{\text{rst}})/t_{\text{task}} \rfloor. \tag{10}$$

If $l > 0$, we schedule the i^{th} checkpointing at $c_i = f_j + t_{\text{rst}} + l \cdot t_{\text{task}}$. Otherwise, no checkpointing is made, as there is not enough time to complete even one task.

IV. Competitive Online Checkpointing Algorithm

In this section, we present a $\frac{1}{2}$-competitive online algorithm with which users can expect *at least* $\frac{1}{2}$ of the effective computation time achieved by the optimal offline solution. We show that this is the best possible among all online algorithms.

A. Competitive Ratio

Following the standard competitive analysis [10], we use *competitive ratio* as the performance metric of our online algorithm. Specifically, given a revocation sequence $F = \{f_1, f_2, \ldots, f_n\}$, let $T_{\text{EC}}^{\text{OPT}}$ be the effective computation time achieved by an optimal offline solution. Let $T_{\text{EC}}^{\mathcal{A}}$ be similarly defined for an online Algorithm $\mathcal{A}$. We say Algorithm $\mathcal{A}$ is α-*competitive* if for all revocation sequence F, the following inequality holds:

$$T_{\text{EC}}^{\mathcal{A}} \geq \alpha T_{\text{EC}}^{\text{OPT}}. \tag{11}$$

The competitive ratio of Algorithm $\mathcal{A}$ is α.

B. Competitive Online Algorithm

Our online algorithm simply checkpoints the output of each task right after its completion. Algorithm 1 gives the details.

Algorithm 1: Online Algorithm $\mathcal{A}$

```
Let c_i be the scheduled time of the i^th checkpointing. Let t be
  the current time. Initially, i ← 1, t ← 0.
Schedule the next checkpointing at c_i ← t + t_rst + t_task.
while true do
    if a revocation occurs then
        c_i ← t + t_rst + t_task.
    end
    if the i^th checkpointing completes successfully then
        i ← i + 1.
        c_i ← t + t_task.
    end
end
```

We show through the following theorem that our online algorithm is $\frac{1}{2}$-competitive.

Theorem 1. *For all revocation sequence, we have*

$$T_{EC}^{\mathcal{A}} \geq \tfrac{1}{2} T_{EC}^{OPT}. \tag{12}$$

Proof. Given an instance revocation sequence $F = \{f_1, f_2, \ldots, f_n\}$, we consider the interval between two consecutive revocations f_j and f_{j+1}, and let δ_j be the time span of the interval, i.e.,

$$\delta_j = f_{j+1} - f_j. \tag{13}$$

Let $T_{\text{interval-}j}^{\mathcal{A}}$ denote the effective task computation time during interval δ_j using our online algorithm. The total effective computation time is simply

$$T_{\text{EC}}^{\mathcal{A}} = \sum_j T_{\text{interval-}j}. \tag{14}$$

Let $T_{\text{interval-}j}^{\text{OPT}}$ and $T_{\text{EC}}^{\text{OPT}}$ be similarly defined for the optimal offline algorithm.

Let $l_{\text{interval-}j}^{\text{OPT}}$ be the number of tasks whose computation results get checkpointed by the optimal offline algorithm at the end of interval δ_j. It is easy to verify that

$$l_{\text{interval-}j}^{\text{OPT}} = \lfloor (\delta_j - t_{\text{ckp}} - t_{\text{rst}})/t_{\text{task}} \rfloor. \tag{15}$$

Let $l_{\text{interval-}j}^{\mathcal{A}}$ be similarly defined for our online algorithm. Since the online algorithm checkpoints upon a task completion, we have

$$l_{\text{interval-}j}^{\mathcal{A}} = \lfloor (\delta_j - t_{\text{rst}})/(t_{\text{task}} + t_{\text{ckp}}) \rfloor. \tag{16}$$

We next show that for each interval δ_j, the online algorithm retains at least half of the effective computation time of the optimal offline solution, i.e.,

$$T_{\text{interval-}j}^{\mathcal{A}} \geq \frac{1}{2} T_{\text{interval-}j}^{\text{OPT}}. \tag{17}$$

We differentiate between the following two cases:

Case-1: $\delta_j < t_{\text{task}} + t_{\text{ckp}} + t_{\text{rst}}$. In this case, the interval between two revocations is too short to checkpoint a task's output. Both algorithms have zero effective computation time:

$$T_{\text{interval-}j}^{\mathcal{A}} = T_{\text{interval-}j}^{\text{OPT}} = 0. \tag{18}$$

Therefore, the statement trivially holds.

Case-2: $\delta_j \geq t_{\text{task}} + t_{\text{ckp}} + t_{\text{rst}}$. We further consider two subcases, depending on whether $l_{\text{interval-}j}^{\text{OPT}}$ is *even* or *odd*. We start by assuming that $l_{\text{interval-}j}^{\text{OPT}}$ is even. Recall that the online algorithm makes checkpointing right after each task completion, and the checkpointing time is less than the task runtime, i.e., $t_{\text{ckp}} \leq t_{\text{task}}$ (cf. Eq. (1)). These immediately suggest that the online algorithm checkpoints the output of at least $\frac{1}{2} l_{\text{interval-}j}^{\text{OPT}}$ tasks in interval δ_j. Therefore, we have

$$\frac{T_{\text{interval-}j}^{\mathcal{A}}}{T_{\text{interval-}j}^{\text{OPT}}} = \frac{l_{\text{interval-}j}^{\mathcal{A}}}{l_{\text{interval-}j}^{\text{OPT}}} \geq \frac{1}{2}, \tag{19}$$

and Eq. (17) holds.

We next assume that $l_{\text{interval-}j}^{\text{OPT}}$ is odd. With similar analysis, we have

$$\frac{T_{\text{interval-}j}^{\mathcal{A}}}{T_{\text{interval-}j}^{\text{OPT}}} = \frac{l_{\text{interval-}j}^{\mathcal{A}}}{l_{\text{interval-}j}^{\text{OPT}}} \geq \frac{1}{2} + \frac{1}{2 l_{\text{interval-}j}^{\text{OPT}}} \geq \frac{1}{2}. \tag{20}$$

From the discussions of the two cases, we see that Eq. (17) holds for all interval δ_j. Based on this result, we establish the theorem as follows:

$$\frac{T_{EC}^{\mathcal{A}}}{T_{EC}^{\text{OPT}}} = \frac{\sum_{j=1}^{n-1} T_{\text{interval-}j}^{\mathcal{A}}}{\sum_{j=1}^{n-1} T_{\text{interval-}j}^{\text{OPT}}} \geq \frac{1}{2}. \tag{21}$$

□

Theorem 1 states that, regardless of the revocations, the effective computation time achieved by our online algorithm is at least half of that of the optimal offline solution. We next show through the following theorem that this is the best possible one can expect without the knowledge of future revocations.

Theorem 2. *Algorithm 1 is the* only *online solution that gives a constant competitive ratio.*

Proof. For any online algorithm other than our solution, say, Algorithm $\mathcal{B}$, we construct a revocation sequence such that Algorithm $\mathcal{B}$ retains zero competitive ratio. In particular, we assume there is an adversary that always schedules an instance revocation right after Algorithm $\mathcal{B}$ starts checkpointing. Because Algorithm $\mathcal{B}$ behaves differently from Algorithm 1, there exists a checkpointing interval in which the checkpointing is made after at least two task completions. Let L denote the time span of this checkpointing interval. We have

$$L \geq 2t_{\text{task}} + t_{\text{rst}} \geq t_{\text{task}} + t_{\text{ckp}} + t_{\text{rst}}. \tag{22}$$

We now turn to the optimal offline algorithm, which checkpoints at least one task's output in L. This suggests that the effective computation time of the optimal offline algorithm is non-zero ($\geq$ one task runtime), i.e., $T_{\text{EC}}^{\text{OPT}} > 0$. As a result, we show that Algorithm $\mathcal{B}$ retains zero competitive ratio: $T_{\text{EC}}^{\mathcal{B}}/T_{\text{EC}}^{\text{OPT}} = 0$. □

Theorem 2 immediately suggests that Algorithm 1 is the optimal online solution with the maximum competitive ratio.

V. Heuristic Online Checkpointing Algorithm

In this section, we present an online heuristic algorithm that achieves better average performance than the competitive solution.

A. Motivation

We have shown previously that to achieve performance guarantee, one has to aggressively checkpoint the output of each task right after its completion. However, this algorithm is too pessimistic about the instance reliability and may incur high checkpointing overhead given infrequent revocations. In an extreme case where the transient server is highly reliable, the algorithm would waste $t_{\text{ckp}}/(t_{\text{ckp}} + t_{\text{task}})$ share of time on checkpointing. The shorter the task is, the more time would be wasted. We expect that this problem would become more salient in the future, given that cloud clusters are shifting towards short-running tasks. To address this problem, we propose a heuristic algorithm that achieves a better average performance by adaptively adjusting the checkpointing frequency.

B. Heuristic Online Algorithm

Our heuristic algorithm dynamically maintains the *checkpointing interval* and updates it in a way similar to the congestion window (`CWND`) updating in the TCP congestion control protocol [11]. In TCP, `CWND` is additively increased by one upon a successful packet delivery and multiplicatively decreased by half in the presence of a packet loss (a.k.a. AIMD). Our heuristic algorithm follows the same intuition. In particular, the checkpointing interval is initially set to one task runtime t_{task}, meaning that the first checkpointing is scheduled right after the first task completion. Whenever a checkpointing has been made successfully, the algorithm conservatively increases the interval by one task runtime. On the other hand, upon a revocation occurrence, the algorithm aggressively reduces the interval by half, with the minimum interval being one task runtime. Algorithm 2 elaborates the details.

Algorithm 2: Heuristic Algorithm $\mathcal{H}$

Let $\text{interval}_{\text{ckp}}$ be the checkpointing interval. Initially, $\text{interval}_{\text{ckp}} \leftarrow t_{\text{task}}$.
Let c_i be the scheduled time of the i^{th} checkpointing. Let t be the current time. Initially, $i \leftarrow 1, t \leftarrow 0$.
Schedule the next checkpointing at $c_i \leftarrow t + t_{\text{rst}} + \text{interval}_{\text{ckp}}$.
while ***true*** **do**
  if *a revocation occurs* **then**
    $\text{interval}_{\text{ckp}} \leftarrow \lceil \frac{\text{interval}_{\text{ckp}}}{2 \cdot t_{\text{task}}} \rceil \cdot t_{\text{task}}$.
    $c_i \leftarrow t + t_{\text{rst}} + \text{interval}_{\text{ckp}}$.
  end
  if *the i^{th} checkpointing completes successfully* **then**
    $i \leftarrow i + 1$.
    $\text{interval}_{\text{ckp}} \leftarrow \text{interval}_{\text{ckp}} + t_{\text{task}}$.
    $c_i \leftarrow t + \text{interval}_{\text{ckp}}$.
  end
end

VI. Evaluation

In this section, we evaluate the performance of the two proposed online algorithms through trace-driven simulations. Simulation results confirm the performance guarantee of our $\frac{1}{2}$-competitive online algorithm. For workloads with short-running tasks, our heuristic algorithm achieves better average performance with shorter job completion time.

A. Simulation Setup

Amazon EC2 Spot Instances: Our simulations are based on Amazon EC2 Spot instances of three Linux types: `c4.large` with high computing performance, `t1.micro` with low price and `m2.xlarge` with highly fluctuated prices. We collected their price history in 88 days from 12/27/2016 to 03/24/2017, based on which the instance revocation sequence can be obtained.

Checkpointing/Restarting Overhead: Prior work reveals that the required checkpointing time is in proportion to an instance's memory footprint [12]. The measurement result in [3] show that checkpointing 240 MB worth of data takes 6.83 seconds. Based on these observations, we assume a constant checkpointing rate of 30 seconds per GB of data in our simulations. For simplicity, we assume that the task restart time is equal to the checkpointing time, i.e., $t_{\text{rst}} = t_{\text{ckp}}$. Table II summarizes the simulation settings.

TABLE II: Simulation settings.

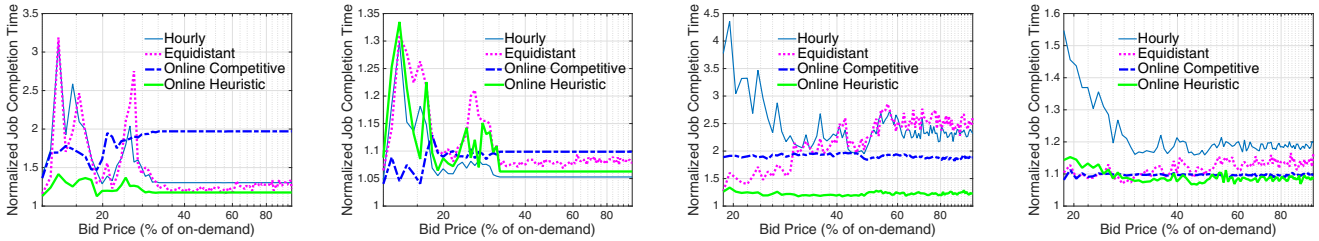
Instance Type	**On-demand Price** ($ per hour)	**Memory** (GB)	t_{ckp} (s)
`c4.large`	0.1	3.75	112.5
`t1.micro`	0.02	0.615	18.5
`m2.xlarge`	0.245	17.1	513

Baselines: We compare our online algorithms against the two baseline alternatives respectively proposed in [4] and [3].

1) *Hourly-boundary checkpointing*: The checkpointing is made periodically in an hourly basis [4].
2) *Optimal equidistant checkpointing*: The algorithm makes checkpointing periodically in a fixed time interval. The checkpointing interval is optimally computed based on the statistical mean time between revocations (MTBF) [3]. In the simulation, the MTBF is computed by randomly sampling the revocation arrivals in 9 consecutive days.

Workload: In our simulations, we assume a cloud job with 100 tasks whose arrival time is uniformly taken in 88 days. We assume two types of tasks: short-running tasks with $t_{\text{task}} = t_{\text{ckp}}$ and long-running tasks with $t_{\text{task}} = 10t_{\text{ckp}}$.

Performance Metric: We measure the performance of a checkpointing algorithm by the *normalized job completion time*, defined as the total instance time used for job execution (excluding the out-of-bid time) under the due algorithm *normalized* by that of the optimal offline solution. The smaller the normalized job completion time, the more cost saving the checkpointing algorithm can achieve.



(a) `c4.large`, $t_{\text{task}} = 1t_{\text{ckp}}$ (b) `c4.large`, $t_{\text{task}} = 10t_{\text{ckp}}$ (c) `t1.micro`, $t_{\text{task}} = 1t_{\text{ckp}}$ (d) `t1.micro`, $t_{\text{task}} = 10t_{\text{ckp}}$

Fig. 3: Job execution time ratio based on the execution time that the optimal offline solution incurs.

B. Simulation Results

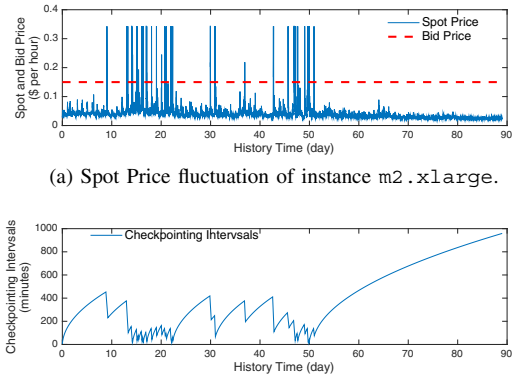
Fig. 3 compares the normalized job completion time under different checkpointing algorithms, where each data point has been averaged over 100 runs.

Competitive Online Algorithm: As expected, our competitive online algorithm achieves guaranteed performance: The normalized job completion time is less than two. This is not possible with the two baseline algorithms. Additionally, we see that the competitive algorithm achieves *near-optimal* performance for long-running tasks. The normalized job completion time is close to 1 in Figs. 3b and 3d, as opposed to Figs. 3a and 3c, where the normalized job completion time is close to 2.

Heuristic Online Algorithm: We now demonstrate the effectiveness of our heuristic algorithm through Figs. 3 and 4.

We see from Figs. 3a and 3c that, for short-running tasks, our heuristic algorithm achieves the best performance with the shortest job completion time among all four solutions.

To better understand the behaviors of our heuristic algorithm, we micro-benchmarked its performance on an `m2.xlarge` instance with bid price 0.15$. Fig. 4a shows the dynamic Spot price over time, and Fig. 4b shows the corresponding checkpointing intervals. We see that when the instance revocation frequently occurs (e.g., during the 15$^{\text{th}}$ and the 25$^{\text{th}}$ days), the heuristic algorithm quickly reacts with a short checkpointing interval. This way, the checkpointing is made more aggressively to counter the frequent work interruptions. Starting the 55$^{\text{th}}$ day, the Spot price becomes consistently lower than the bid price. The heuristic algorithm reacts with a longer checkpointing interval, saving more time for computation.



(a) Spot Price fluctuation of instance `m2.xlarge`.

(b) Dynamic Variation of the Checkpoint Intervals.

Fig. 4: Effectiveness of the Heuristic Algorithm.

VII. Concluding Remark

In this paper, we proposed two online checkpointing strategies, one competitive and the other heuristic, for cloud transient servers without prior knowledge of future instance revocations. We showed that our competitive algorithm achieves the best possible competitive ratio $\frac{1}{2}$. Our heuristic algorithm, on the other hand, achieves better average performance, especially for workloads with short-running tasks. Trace-driven simulations based on Amazon EC2 Spot price history confirm the performance guarantee of the competitive online algorithm and demonstrate the effectiveness of the heuristic algorithm.

References

[1] Amazon EC2 Spot Instances. [Online]. Available: https://aws.amazon.com/ec2/spot/

[2] Preemptible VM Instances. [Online]. Available: https://cloud.google.com/compute/docs/instances/preemptible

[3] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, and F. Cappello, "Optimization of cloud task processing with checkpoint-restart mechanism," in *Proc. IEEE/ACM Intl. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[4] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of Spot Instances via checkpointing in the Amazon Elastic Compute Cloud," in *Proc. IEEE CLOUD*, 2010.

[5] D. Jung, S. Chin, K. Chung, H. Yu, and J. Gil, "An efficient checkpointing scheme using price history of Spot Instances in cloud computing environment," in *IFIP Intl. Conf. Netw. Parallel Comput.*, 2011.

[6] Y. Gong, B. He, and A. C. Zhou, "Monetary cost optimizations for mpi-based hpc applications on amazon clouds: Checkpoints and replicated execution," in *Proc. IEEE/ACM Intl. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, 2015.

[7] B. Javadi, R. K. Thulasiramy, and R. Buyya, "Statistical modeling of spot instance prices in public cloud environments," in *Proc. IEEE Intl. Conf. Utility and Cloud Computing (UCC)*, 2011.

[8] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. N. Tantawi, and C. Krintz, "See Spot run: Using Spot Instances for MapReduce workflows," in *Proc. USENIX HotCloud*, 2010.

[9] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing Amazon EC2 Spot Instance pricing," *ACM Trans. Econ. Comput.*, vol. 1, no. 3, pp. 16:1–16:20, 2013.

[10] A. Borodin, N. Linial, and M. E. Saks, "An optimal on-line algorithm for metrical task system," *J. ACM*, vol. 39, no. 4, pp. 745–763, 1992.

[11] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, 1988.

[12] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Resource leasing and the art of suspending virtual machines," in *Proc. IEEE Intl. Conf. High Perform. Comput. Commun.*, 2009.