Backup or Not: An Online Cost Optimal Algorithm for Data Analysis Jobs Using Spot Instances

LIDUO LIN, LI PAN, (Member, IEEE), AND SHIJUN LIU, (Senior Member, IEEE)
School of Software, Shandong University, Jinan 250101, China

Corresponding authors: Li Pan (panli@sdu.edu.cn) and Shijun Liu (lsj@sdu.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2017YFA0700601.

ABSTRACT Recently, large-scale public cloud providers begin to offer spot instances. This type of instance has become popular with more and more cloud users in the light of its convenient access mode and low price, especially for those big data analysis jobs with high performance computation requirements. However, using spot instances may carry the risk of being interrupted and lead to extra costs for job re-executions because these instances are generally unstable. Yet, such cost can be greatly reduced if a backup can be made at the right time before interruptions. For convenience and cost efficiency, users can choose the StaaS (Storage-as-a-Service) storage provided by the same cloud provider, whose spot instances are used by the users, to store backup data files for future job execution recovery. Since making backups too often will incur increased costs, users need to make the backup decisions appropriately considering the condition when an abrupt interruption will occur in the future. However, it is hard to know or predict precisely when such an interruption will occur. For solving this problem, in this article, we propose an online algorithm to guide cloud users to make backups when using spot instances to execute big data analysis jobs, without requiring any information about future interruptions. We prove theoretically that our proposed online algorithm can guarantee a bounded competitive ratio less than 2. Finally, according to extensive experiments, we verify the effectiveness of our online algorithm in reducing the additional cost caused by interruptions in using spot instances and find that our online algorithm can still achieve a stable cost optimization even if interruptions occur frequently.

INDEX TERMS Spot instance, online algorithm, back up, abrupt termination.

I. INTRODUCTION

The Data Computing and Hosting Services Market is expected to register a CAGR (Compound Annual Growth Rate) of over 8.7% during the forecast period 2020 – 2025 [1]. Companies in emerging economies are increasingly outsourcing their IT infrastructure needs, which further promotes the business growth of the IaaS (Infrastructure-as-a-Service) market. These large-scale public IaaS cloud providers like Amazon, Microsoft or Google offer multiple instance purchasing options including on-demand instances, reserved instances and spot instances. Transferring Hadoop [2], MapReduce [3], Spark [4] and other frameworks from the local to the cloud has become a choice for many data analysis users. In the face of high fees of using high performance on-demand and reserved instances, using spot instances becomes a popular option for cloud users to reduce their IaaS spendings.

Spot instances were firstly available to users by Amazon in late 2009, which were called Amazon EC2 Spot Instances (SI) [5]. This type of instance offers an ultra-low discount compared to the price of corresponding on-demand instances. Users can save 90% of the price of an on-demand instance by selecting "spot" when starting an EC2 instance. Yet it carries the risk of being interrupted because the price and availability of these instances will change dynamically with the supply and demand of instance markets. To use spot instances effectively, users must carefully weigh the low cost against poor availability. Taking Amazon EC2 as an example, the price of spot instances are set by Amazon and gradually adjusted according to the long-term supply and demand of spot instances. Thus, the price of spot instances can fluctuate up and down. A maximum bid needs to be given before a user rents spot instances. Such a bidding
strategy is called “Set your max price (per instance/hour)”. Generally, users will estimate the price and weigh reliability against the price themselves. A user’s job will keep running in spot instances whenever the maximum bid, which is set by the user, exceeds the current price of the spot instances. Once the current price of the spot instances are higher than the maximum bid, the user’s job will be terminated and irrecoverable. Despite their low reliability, many case studies show that spot instances have huge potential in rendering on the cloud, machine learning, big data analysis and some other batch processing fields [6]. These jobs are generally compute intensive and cost much to run, but often having high flexibility and not latency critical [7]. In these situations, users would rather have these jobs delayed a little than pay high fees.

Therefore, there is a trade-off between cost and reliability of using spot instances. The cost management of using spot instances has been studied by a lot of work. There exists some work [8]–[10] aiming at optimizing users’ bids for cost management. However, they heavily rely on long-term predictions, while price forecasts are sometimes unreliable, especially when overwhelming demands are generated. Improving the reliability with scheduling algorithms by combining different pricing models (spot and on-demand instances) [11] and different available regions [12] have also been mentioned. In addition to the above approaches, major cloud providers such as AWS [13] advise users to make backups when they choose spot instances.

Using appropriate checkpoints can balance cost and reliability. For example, a lot of big data analysis jobs are carried out in stages and produce intermediate data at every stage, which can be used as a checkpoint for recovery or secondary analysis. Without any backup, once an interruption occurs, with the right of use of the instances being revoked, the previous job executions will be lost and the interrupted job needs to be re-executed from the beginning, which causes economic and time losses. If the intermediate data are stored in advance, a job can be recovered from the last checkpoint after an interruption.

Since most IaaS cloud providers offer computing capacity as well as data storage service, which is called Storage-as-a-Service (StaaS) such as Amazon Simple Storage Service (Amazon S3) [14] and Google Cloud Storage [15], users can choose to store the intermediate data on StaaS storage provided by the cloud provider of the spot instances they choose. Virtually unlimited cloud storage while saving maintenance costs by moving data from internal physical servers to cloud storage is very attractive to users. In that case, network traffic costs will account for a significant portion of these StaaS services. The spending of a single backup usually also costs much, which means that we can’t do it blindly. For example, we consider a big data analysis job running on a distributed system which has a master and several slaves in different regions. Suppose this job occupies 10 GB kept in S3. Then, we calculate the cost of a single backup as: S3 request fee + network transfer fee = (0.005+0.02*10)

### TABLE 1. AWS EC2 instance price in January 2020.

| Instance type | vCPU | Memory (GB) | Spot (per hour) | On-Demand (per hour) |
|---------------|------|-------------|------------------|----------------------|
| a1-medium     | 1    | 2           | $0.0084          | $0.0255              |
| a1.xlarge     | 4    | 8           | $0.0197          | $0.1020              |
| m4-10xlarge   | 40   | 160         | $0.6800          | $2.0000              |
| m5n-24xlarge  | 96   | 384         | $1.6323          | $5.7412              |

USD = 0.205 USD [14]. We can see this cost is much higher than the instance running costs shown in Table 1. Thus, an arbitrary backup decision can be expensive.

The problem now confronting users lies in that, for making cost-efficient backup decisions, users need to know exactly when an interruption will occur, which is generally very difficult to predict. In order to solve the above problems, in our work, we propose to use online algorithms to guide cloud users to decide whether and when to back up current job executions with no a priori knowledge of future abrupt termination occurrence distribution. Specifically, in this article, we use big data analysis jobs running in AWS Spot Instance and AWS S3 StaaS cloud as an example to illustrate our algorithm. We design an online algorithm to determine when to back up the intermediate results of data analysis jobs to StaaS and theoretically prove that our online algorithm can obtain guaranteed competitiveness in terms of saving the cost of running data analysis jobs on spot instances. Finally, through a large number of experimental simulations, we verify the efficiency of our algorithm.

The rest of the paper is organized as follows. Section II gives a brief introduction of the related work. In Section III, we describe our system and cost model. Section IV presents our online algorithm and analyzes its performance. Experimental simulations are involved in Section V. Finally, we draw conclusions and propose future work directions in Section VI.

### II. RELATED WORK

We firstly start with the work related to cost optimization problems for cloud users. Chaisiri et al. [16] show an optimal cloud resource provisioning algorithm for cloud computing which uses more costly reservation instances and less costly on-demand ones. Chiaraviglio et al. [17] design an algorithm (MECDC) to solve the problem of managing the power states of the servers in a Cloud Data Center (CDC) for saving cost. Wu et al. [18] propose an algorithm which takes both data transmission cost and computation cost into account. In fact, cloud disaster is very common in cloud environment and current disaster recovery services provided by cloud providers come at very high cost [19]. Many work hopes to avoid this part of the cost through the self-disaster warning and recovery. Eric et al. [20] devise a cloud migration system to achieve fast workload transition in case of disaster. Gunawi et al. [21] propose a new testing framework for cloud recovery so that the loss of cloud disaster can be reduced. The above articles are all proposed for on-demand instances or reserved instances which have very different characteristics compared with spot instances. In this article, we deal with the cost optimization problems for spot instances.
Spot instances are playing an importance role in current instance markets. A lot of work is devoted to analyze the spot instance environments and help users better use spot instances by cost management. Singh et al. [22] take spot instance as template and design Yank with a bounded-time VM migration mechanism to exploit the advance warning, which dramatically decreases the cost of providing high availability relative to existing solutions. Ben-Yehuda et al. [23] analyze the spot price histories of Amazon’s EC2 cloud. They construct a model and reversely engineer how prices are set and find the price may be generated most of the time at random from within a tight price range via a dynamic hidden reserve price mechanism. Fabra et al. [24] design a framework which can classify the spot instance availability zones and then generate price prediction models adapted to each class for generating resource provisioning plans that get the optimal price. The bidding strategy based on online algorithm [25] is also proved by Guo et al. to be able to achieve price optimization. They use various methods to optimize the cost. In this work, we will propose a backup strategy for cost management.

The use of appropriate checkpoints can realize cost optimization when using spot instances. Such checkpointing strategies are introduced in [26]–[28], which are based on the prediction of future interrupt distribution. However, accurate prediction is very difficult. Our proposed algorithm is closely related to the research of online algorithms [29] which don’t require any a priori knowledge of future. Our backup problem can be seen as an analogy to the classical rent-or-buy problems including Bahncard problem [30], ski board rental problem [31], TCP acknowledgment problem [32] and so on.

Based on the above discussion, our main contribution lies in that we design an online algorithm and prove its effectiveness for cost management in using spot instances.

III. SYSTEM AND COST MODEL
A. SYSTEM MODEL
We consider a system which runs big data analysis application jobs based on IaaS and StaaS cloud platforms to illustrate our algorithm. Fig. 1 shows our system, which is built on AWS EC2 Spot Instance cluster and S3 standard storage service. Raw data are collected from various sources, and then analyzed by a spot instance cluster which is a virtualized computing cluster composed of multiple spot instances. The spot instances in a cluster interact with each other and
transmit data through network. Each data analysis job runs in parallel on a spot instance cluster and its generated data can be stored in S3 if a backup decision is made. A data analysis job is usually conducted in multiple stages, and intermediate results will be generated in each stage, which can be used for secondary analysis or checkpoint recovery.

Fig. 1 also shows the working principle of our system. Our system can be described as a certain web service, which can be invoked by a user with a data analysis job to be executed in clouds. When running a user’s job, our system will decide when and whether to back up the intermediate results and then store intermediate data in a fixed storage space that the user rents. This process will be carried out in two steps to prevent the interruptions during a backup. First, the data are transmitted by network and stored in the buffer, which is a temporary block of the same size as the fixed storage space. And then, data in the buffer are flushed to storage only when this backup task is completed. Finally, the buffer is released.

### B. FUNDAMENTAL NOTATIONS

For Amazon cloud platform, users are charged by hours and each user sets the max bid itself. Running spot instances has the risk of interruptions. In general, when the current price is lower than the max bid, a job will keep running normally on spot instances. Otherwise, when the current price is higher than the max bid, the right of use of the spot instances will be revoked by the provider and the user’s job will be suspended. The user’s job will be lost permanently and irrecoverably. When the current price falls again below the user’s max bid, the user can regain the access to instances and continue to run the job after a checkpoint recovery. If the job hasn’t been backed up before, the system needs to restart the job execution from the beginning. If the job has made a backup at some point before the interruption, the job can restart from the latest backup point. Backup costs a certain amount of money. An arbitrary backup decision can be costly. In order to optimize the cost spent in cloud platform, users need to decide when and whether to make backups.

In this article, we deal with the situation that users run data analysis jobs which will produce intermediate results in our system. The interruptions can be caused by price changes or insufficient instance resources. The latter is less likely to happen, but the consequences are the same as the former. We only consider the interruptions caused by price changes in the subsequent derivation. This will not affect the final experimental results. For the convenience of follow-up reference, we list the symbols used in this article in Table 2.

### C. STORAGE COST

As mentioned earlier, a running job will produce intermediate results and store them in S3 cloud storage. Even if the system does not have abrupt termination, the storage cost is inevitable because raw data storage is also required. We denote the size of intermediate results as $S_b$ and the cost of storage for S3 per hour as $P_s$. Then the storage cost can be calculated as a linear function of time $t$:

$$C_{storage}(t) = S_b P_s t.$$  

(1)

It is also mentioned above that the size of storage space in StaaS is fixed. If the intermediate data stored in StaaS is only used for checkpoint recovery, each backup can overwrite the last backup to minimize storage cost. Suppose a storage space whose size reaches 10 GB kept in S3 and the completion time for a job running without abrupt termination is 24 hours. $S_b$ is about 0.022 USD per GB per month. Then, the cost of occupying this space incurs a charge of: $10/GB \times 24h \times 0.002/month/GB/30 = $0.0073 \text{ (per day)} = $0.0003 \text{ (per hour)}$.

### D. DATA COMPUTATION COST

Now we consider a data analysis job running in our system. A user selects different instances to form the entire computing cluster. Amazon charges by hours. The price of the instance cluster is determined by the sum of the unit price corresponding to a single instance and denoted as $C_r$. $C_r$ will fluctuate from time to time. Let $C_r(t)$ denote the price of the cluster in time $t$, and $C_m$ denote the user’s max bid. When $C_r(t) > C_m$, the job will be interrupted and should wait for the price to fall. The cost from start to time $t$ can be calculated as:

$$C_{computation} = \sum_{i=0}^{t} C_r(i).$$  

(2)

### E. BACKUP COST

The cost of backup includes the cost of storage requests and network transmission. All operations on the cloud storage platform are ultimately calls to the platform API. Each backup and recovery is the request calls to S3 from each instance in the cluster. So, this part of cost is also constant every time and denoted as $P_b$. During data transmission, we request a temporary space in S3 for buffering. After data transmission, it will import all the data into the fixed storage space. It will then be released immediately. We denote the cost of requesting a buffer in S3 as $P_r$. Network transfer cost is the multiplication of network transfer fee per GB and the size of intermediate results. Thus, we calculate the cost of network transfer for a single backup as: $S_b P_r$.

### TABLE 2. The symbols used in this article.

| Symbol | Meaning |
|--------|---------|
| $T$    | the completion time for a job running without abrupt termination |
| $n$    | the number of abrupt terminations |
| $C_r$  | the cost of spot instance cluster per hour |
| $C_m$  | the max bid for the instance cluster |
| $P_b$  | the price of requesting calls to S3 |
| $P_r$  | the price of requesting a buffer in S3 |
| $S_b$  | the size of intermediate results |
| $P_n$  | the price of network transfer per GB |
| $P_s$  | the price of storage for S3 per hour |
| $\tau$ | break-even point |
Backup cost for an object can be calculated as:

\[ C_{\text{backup}} = P_q + P_b + S_b P_n. \]  

(3)

### F. RECOVERY COST

When an interruption occurs, the right of use of the instances in the cluster will be recalled and all data in the cluster will not be kept. When the job is restarted after an interruption, the instance cluster must reload the data set from S3. The cost of this part is basically the same as the backup cost without the cost of requesting a buffer in S3. Backup is for uploading the data to S3, and recovery is for downloading it. So the difference between the two is only reflected in the network cost of uploading and downloading, and in most cases they’re pretty much the same. For the convenience of calculation, we denote the cost of recovery as:

\[ C_{\text{recovery}} = P_q + S_b P_n. \]  

(4)

### G. THE COST OPTIMIZATION PROBLEM

As introduced earlier, we use StaaS storage to store intermediate data for checkpoint recovery. If we make backups too often, it will incur unnecessary extra cost. If we do not make a backup for a too long period, the risk of cost caused by an abrupt termination will continue to increase. When a job keeps running, we need to decide whether and when to make backups. An appropriate backup time point should be determined to optimize the cost and avoid re-executing from the beginning for saving execution time.

Considering a job, if we can know the exact value of future prices of spot instances, it is feasible to look at the big picture for decisions that minimize costs. We can then decide either to make a backup before the price exceeds the max bid or to give up because making a backup may cost more. However, the future abrupt termination occurrence distribution is generally hard to predict for cloud users. Hence, backup decisions should be made online, without requiring any a priori knowledge of future interruption occurrence distribution.

### IV. ONLINE COST OPTIMIZATION ALGORITHM

In this section, we firstly discuss the problem of online cost optimization and the challenges in selecting backup decision points. Then we design a new online cost optimization algorithm and theoretically analyze its performance.

#### A. ONLINE COST OPTIMIZATION PROBLEM

In this work, what the online cost optimization problem considers is to make a decision on when to make a backup without any a priori knowledge of future price changes. The key step in designing an online algorithm is to find the break-even point denoted as \( \tau \), which is in fact a time between two backups. At the break-even point, the cost incurred by re-executing from the last backup point is the same as making a backup. Next, we show how to calculate the break-even point.

The cost incurred by re-execution is determined by the future average price of the instance cluster, which can only be estimated. As shown in Fig. 2, we use \( t \) to denote the current time and \( \Delta t \) to denote the time difference from the last backup point. The period of the time between \( (t - \Delta t, t] \) is denoted as the estimation window. The period of the time from the beginning of the re-execution to the next backup is denoted as the event window\(^1\). The upper limit of the price in event window is the max bid of the user and the lower limit is the lowest price of the instance cluster. The average of historical prices in the estimation window is within this range and represents the recent trend of the prices of the instance cluster. We use the average price in the estimation window to estimate the average price in the event window. As an estimation of the future average price of the instance cluster, the historical average price is denoted as:

\[ C_{\text{avg}} = \frac{1}{t} \sum_{i=t-\Delta t}^{t} C_i(t). \]  

(5)

As mentioned above, a single backup can be described as (3). We denote the cost of re-execution from last backup point as:

\[ C_{\text{re-execution}} = (S_b P_s + C_{\text{avg}}) \Delta t. \]  

(6)

Then we calculate the break-even point by solving the equation of \( C_{\text{backup}} = C_{\text{re-execution}} \) as follows:

\[ P_q + P_b + S_b P_n = (S_b P_s + C_{\text{avg}}) \Delta t \]

\[ \rightarrow \tau = \Delta t = \frac{P_q + P_b + S_b P_n}{C_{\text{avg}} + S_b P_s}. \]  

(7)

### B. DESCRIPTION FOR OUR ONLINE ALGORITHM

The operation principle of our online algorithm is summarized as follows. The raw data is stored in StaaS and the instance cluster for running a job is then started. Our algorithm records the time duration from the last backup point as \( \Delta t \). It determines whether a backup is needed by hours according to the break-even point \( \tau \). The job will continue to run if \( \Delta t < \tau \). When \( \Delta t \geq \tau \), our algorithm decides to make a backup and marks this time as the nearest backup point. Once the instance cluster price exceeds the maximum bid, the right of use of the instance cluster will be revoked and the user’s job should wait for the price to fall. When the price falls below the maximum bid, the job needs to continue running from the last backup point. Our online algorithm makes decisions

---

\(^1\)In event analysis, the whole event is divided into estimation window and event window, and the statistical characteristics of estimation window are used as the estimation of event window. This form of analysis has been widely used in the fields of financial modeling [33], disease prediction [34], computer science [35] and so on.
Algorithm 1 An Online Backup Algorithm for Cost Optimization

1. Let $C_A$ be the total cost of running the user’s job. Initially, $C_A=0$.
2. Let $C_i$ be the price of instance cluster at time $t_i$. Initially, $i \leftarrow 0$ for all $i = 0, 1,...$
3. Let $t_b$ be the last backup point and $t_0$ be the initial backup point, and $t_0 = t_b$.
4. Let $C_m$ be the max bid, $C_{sum}$ be the sum of the prices over a period of time and $C_{avg}$ be the average price over a period of time.

while not at the end of this job do

if $C_i > C_m$ then

| An interruption occurs. |
|-------------------------|
| while True do |
| $i \leftarrow i + 1$. |
| if $C_m \geq C_i$ then |
| $C_A= C_A +$ recovery cost. |
| Break. |
| end |

end

end

Record the time duration between the last backup point: $\Delta t = t_i - t_b$.
Calculate the historical average price:

$$C_{sum} = C_{sum} + C_i; \quad C_{avg} = \frac{C_{sum}}{\Delta t}.$$  

Calculate the break-even point:

$$\tau = \frac{P_q + P_b + S_b P_n}{C_{avg} + S_b P_s}.$$  

if $\Delta t \geq \tau$ then

| Make a backup and let $t_b \leftarrow t_i, C_{sum} \leftarrow 0$. |
| $C_A = C_A +$ backup cost. |
| $i \leftarrow i + 1$. |

end

end

The performance of an online algorithm is usually measured by the ratio of its performance compared to an optimal offline algorithm, denoted as $OPT$, that is well aware of future events. The maximum of this ratio is measured as the competitive ratio of this online algorithm. The value of competitive ratio is always greater than one and the closer to one, the better the online algorithm is. According to the definition of calculating competitive ratio just given, we use the worst case analysis to analyze it [36]. The worst case analysis is to analyze the worst case of an online algorithm and calculate the competitive ratio in this worst case. Then we conclude that our algorithm can achieve a guaranteed competitive ratio.

We denote $C_A$ as the total cost of Algorithm 1 and $C_{OPT}$ as the total cost of algorithm $OPT$. From the definition of competitive ratio, the value of $C_{OPT}$ is always less than or equal to $C_A$. For their consistency, the two backup algorithms should run with the same price changing. In the worst case of Algorithm 1, the instance cluster’s price, $C_r$, will be stable at the max bid except when an interruption occurs. Thus, the break-even point is a fixed value:

$$\tau = \frac{P_q + P_b + S_b P_n}{C_r + S_b P_s}.$$  

In this situation, backups are more frequent and running costs are higher which makes our online algorithm cost more.

We then make an observation that, all interruptions occur as follows. Under normal circumstances, a backup is scheduled if a job runs from $t' - \tau$ to $t'$ and now at the break-even point. At this point, the price exceeds the maximum bid and the re-execution cost is firstly higher than the backup cost. Since the optimal algorithm knows an interruption is about to happen at the time of the break-even point, a backup has completed at the moment before that. But Algorithm 1 does not predict that and thus it hasn’t made a backup promptly. Therefore, it’s the worst case to be interrupted just before it’s about to make a backup because the re-execution cost that Algorithm 1 needs to pay is the largest at this point.

Let $T$ be the completion time for a job running algorithm $OPT$ while $\alpha$ be the number of interruptions. $T$ can be divided into $\beta$ backup point intervals and expressed as:

$$T = \beta \tau + t \quad (0 \leq t \leq \tau; \quad 0 \leq \beta)$$  

As mentioned above, we set the interruptions at each break-even point to achieve the worst case. If the number of interruptions exceeds $\beta$, there will be $\alpha - \beta$ interruptions that have no effect on the competitive ratio calculation according to the Pigeonhole principle\(^2\). In this case, the upper limit of $\alpha$ is $\beta$. If the interruptions happen in a row, the optimal algorithm will be finished in $T$ while Algorithm 1 needs to run from scratch because of no backup. The completion time

\(^2\)[https://en.wikipedia.org/wiki/Pigeonhole_principle]
The meaning of each item of equation (12) is listed in Table 4.

| Item                          | Cost of item                                                                 |
|-------------------------------|-------------------------------------------------------------------------------|
| \(TC_t\)                      | cost of renting instance cluster                                             |
| \(\frac{\beta + \alpha}{\alpha}(P_q + S_bP_n)\) | cost of \(\beta\) times of backup and \(\alpha\) times recovery              |
| \(T + T_w\) \(S_bP_s\)       | cost of storage                                                               |

The meaning of each item of equation (11) is listed in Table 3.

| Item                          | Cost of item                                                                 |
|-------------------------------|-------------------------------------------------------------------------------|
| \(TC_t\)                      | cost of renting instance cluster                                             |
| \(2\alpha(P_q + S_bP_n)\)     | cost of \(\alpha\) times of backup and recovery                              |
| \(T + T_w\) \(S_bP_s\)       | cost of storage                                                               |

for a job with \(\alpha\) times of interruptions in the worst case can be expressed as:

\[ T' = T + \alpha \tau \]  

(10)

After an interruption, the price may not immediately fall back. Users need to pay storage fee when their jobs are stopped and waiting to be resumed. We denote the total waiting time as \(T_w\). Therefore, we set the running time of the instance cluster running algorithm OPT as \(T\), and the storage time as \(T + T_w\). Similarly, the running time of Algorithm 1 can be denoted as \(T'\) and the storage time can be denoted as \(T' + T_w\).

Thus, we have reached the worst case. In this case, \(C_{OPT}\) can be calculated as:

\[ C_{OPT} = TC_t + 2\alpha(P_q + S_bP_n) + \alpha P_b + (T + T_w) S_bP_s \]  

(11)

The meaning of each item of equation (11) is listed in Table 3.

\[ C_A = T'C_r + (\beta + \alpha)(P_q + S_bP_n) + \beta P_b + (T' + T_w) S_bP_n \]  

(12)

The meaning of each item of equation (12) is listed in Table 4.

Hence, we calculate the ratio of \(C_A\) and \(C_{OPT}\) as:

\[
\frac{C_A}{C_{OPT}} = \frac{T'C_r + (\beta + \alpha)(P_q + S_bP_n) + \beta P_b + (T' + T_w) S_bP_n}{TC_r + 2\alpha(P_q + S_bP_n) + \alpha P_b + (T + T_w) S_bP_s} = 1 + \frac{\alpha T C_r + (\beta - \alpha)(P_q + S_bP_n) + (\beta - \alpha) P_b + \alpha \tau S_bP_n}{TC_r + 2\alpha(P_q + S_bP_n) + \alpha P_b + T S_bP_s + T \tau S_bP_s} \leq 1 + \frac{\alpha T C_r + (\frac{T}{\tau} - \alpha)(P_q + P_b + S_bP_n) + \alpha \tau S_bP_n}{T C_r + 2\alpha(P_q + S_bP_n) + \alpha P_b + T S_bP_s + \frac{T}{\tau} S_bP_s} = 1 + \frac{\alpha(T C_r + S_bP_n) + (\frac{T}{\tau} - \alpha)(P_q + P_b + S_bP_n)}{T C_r + 2\alpha(P_q + S_bP_n) + 2\alpha P_q + \alpha P_b} \leq 2 \]  

(15)

\[
\frac{C_A}{C_{OPT}} = 1 + \frac{T}{\tau} + \alpha(1 + \frac{P_q + S_bP_n}{P_q + P_b + S_bP_n}) \leq 2 \]  

(16)

\[
\frac{C_A}{C_{OPT}} = 1 + \frac{T}{\tau} + \alpha \leq 2 \]  

(17)

According to equation (10), we expand the molecule of equation (13) and turn it into the form of \((1 + \chi)\), which are shown as equation (14). Based on equation (9), we get \(\beta \leq \frac{T}{\tau}\) and then we put it into inequation (15) for inequality amplification. In this way, we can deduce equation (16). With equation (8), we use \((P_q + P_b + S_bP_n)\) to replace \((C_s + S_bP_n)\) in equation (17) and get equation (18). In equation (20) we derive the final result.

As the proof shown above, we come to a conclusion that our online algorithm has a good performance as it can reach a competitive ratio less than 2, which means that running our algorithm costs no more than twice as much as the optimal solution.

V. EXPERIMENTAL EVALUATION

In this section, we evaluate our online algorithm’s performance for practical cloud users via a large volume of real-world data sets. In this section we use Online-Spot to denote our online algorithm Algorithm 1 and use Offline to denote the optimal offline algorithm.

A. DATASET DESCRIPTION AND PROCESSING

1) DATA DESCRIPTION

AWS has published the historical prices of its spot instances for the last three months. We collected more than 2000 pieces of data from December 28, 2019 to March 23, 2020 for simulation experiments. This data set can in turn be divided into sub-data sets that are differentiated by instance types, available regions and operating systems. We can use this data set for simulation experiments by running different algorithms to compare their performance in the context of real-world market price fluctuations. Fig. 4 illustrates the market price fluctuations which are randomly selected in this data set.

2) DATA PROCESSING

Since the AWS spot history prices span three months, we proportionally shorten the billing cycle to one hour. To analyze the performance of Online-Spot algorithm under different instance market conditions, we divide all the 2354 data records into three groups according to their price fluctuations levels which are measured as the ratio between
the mean $\mu$ and standard deviation $\sigma$. The results of each category are shown in Fig. 5 and their proportion is shown in Fig. 6.

We use class_1, class_2 and class_3 to distinguish different data categories. Specifically, class_1 consists of the instances whose prices are highly fluctuating, with $\mu < 5\sigma$. It means that these instances usually have dramatic change of market demand and are more likely to be interrupted. We use class_3 to represent the instances whose prices have not changed at all. The price of such instances is very stable and there is no need to make backups. class_2 represents the most common case in the instance market. We can see in Fig. 6 that the size of class_3 is larger than class_1 and class_2. Many points in this class are concentrated near the origin and the points closed to each other are removed to avoid redundancy.

3) USER DATA

We look at the case study provided by AWS [6]. Most users who choose spot instances may pull TB or larger files from Amazon S3, run calculation job and push the results back to S3. They may spin up thousands of spot instances for calculation. So using this situation as a reference, in this experiment we start 2000 instances at a time and a single data backup size of 1PB. Depending on equation (10) we can arrive at a conclusion that our Online-Spot algorithm takes longer to run a job compared to Offline algorithms. We can easily get the effective time ($T$) by running Online-Spot algorithm on each data set and use it to simulate the completion time of a user’s job.

4) PRICING

We adopt the pricing of Amazon EC2 (multi-region) and S3 with the network transfer fee 0.02 USD per GB and storage fee 0.0003 USD per GB per hour.

5) BIDDING STRATEGY

As mentioned above, the interruptions caused by price factors and non price factors have the same influence on our online algorithm. To balance the impact of interruptions caused by price and non price factors on the algorithm, we set the highest bid to no more than twice the lowest price in history (20% of “On-Demand price”). In reality, users should choose the bid according to their own needs, which is not in the scope of our discussion.

B. COMPARISONS WITH USER’S MOST FREQUENTLY USED STRATEGIES

In this subsection, we focus on evaluating the performance of our Online-Spot algorithm, Offline optimal algorithm and some commonly used strategies in this real-world data set.

1) BENCHMARK ALGORITHMS

We compare our proposed Online-Spot algorithm with three benchmark algorithms. The first benchmark algorithm is All-on-Demand, in which the user never needs to make backup decisions because using on-demand instances will not cause interruptions. This strategy is the most common strategy in practice, especially for short term workload. Though simple and stable, the price of on-demand instances is usually about ten times higher than that of spot instances. The second algorithm is called Spot-to-on-Demand. It is a simple extension to the All-on-Demand algorithms. Once an interruption occurs, at the next time when the job restarts, users will choose the corresponding on-demand instances instead. The third benchmark algorithm, Offline, is the optimal algorithm that has a priori knowledge of price fluctuations and makes a decision on whether to make a backup immediately before an interruption. All three benchmark algorithms as well as our online algorithm, are carried out for each historical price trace of the AWS spot instances. All the incurred costs are normalized to All-on-Demand.
2) COST PERFORMANCE EVALUATION

We plot the average ratio of cost savings of all algorithms in Fig. 11 and CDF (Cumulative Distribution Function) in Figs. 7, 8, 9 and 10. They are all grouped by different price fluctuations levels and set algorithm \textit{All-on-Demand} as normalization. Firstly, we see Fig. 7 which represent a stable market environment. \textit{Spot-to-on-Demand} algorithm and \textit{Offline} algorithm have the same performance in this situation. But our \textit{Online-Spot} algorithm costs more for ‘useless’ backups. Those two algorithms can save 10% more than \textit{Online-Spot} algorithm and 80% less than normalized algorithm. Since the price does not change, this strategy can enjoy the price preference by the spot instance without considering the risk brought by interruptions. In this situation our \textit{Online-Spot} algorithm is not as cost-effective as these two algorithms. Fig. 9 shows a situation where price changes are relatively stable. All three algorithms can save the computation cost. \textit{Online-Spot} algorithm costs 60% of the \textit{Offline} optimal solution. \textit{Spot-to-on-Demand} can also get a certain cost advantage. Overall, it saves more than 30% cost compared with the normalized algorithm. But in 60% data cases, the cost is higher than the normalized algorithm. However, when the price fluctuates dramatically as shown in Fig. 8, our algorithm shows its advantages. \textit{Online-Spot} algorithm achieves satisfactory cost-savings by backing up at the right time. It can still guarantee a stable competitive ratio compared with the \textit{Offline} algorithm. As prices fluctuate more and more dramatically the performance of \textit{Spot-to-on-Demand} algorithm becomes worse and worse. As shown in Fig. 8, no more than 10% of the data running the \textit{Spot-to-on-Demand} algorithm cut their costs. The total cost even exceeds 10% of the normalized algorithm. Fig. 10 integrates all the data to get the final result of the three algorithms. We can see that all the three algorithms can achieve cost savings. The cost savings of \textit{Online-Spot} algorithm are more than half the \textit{Offline} algorithm which can confirm with the competitive ratio that we have derived theoretically. As we can observe from Fig. 11, a more intelligent algorithm is essential to prevent the extra cost of interruptions when using spot
instances. No backup preparation like Spot-to-on-Demand or simply using high price on-demand instance will easily cause skyrocketing cost. Our Online-Spot algorithm provides a solution that can run stably in the above environments.

C. COMPARISONS WITH OTHER BACKUP ALGORITHMS

In this subsection, we focus on evaluating the performance of our Online-Spot algorithm by comparing with other benchmark algorithms, including Offline optimal algorithm and some backup algorithms for spot instance cluster proposed by other scholars.

1) BENCHMARK ALGORITHMS

The first benchmark algorithm is called Hour-Boundary, which is a straightforward algorithm. This algorithm makes use of the feature that an hour is the lowest granularity of pricing of spot instances in Amazon EC2. When using this backup strategy, users don’t need to pay attention to other details. Instead, they just need to back up each hour. The second algorithm is called Rising-Edge-Driven which makes backup decisions based on the rising edge of price changes. A rising edge of price changes is likely to indicate that the system has less available resources, more bidding users, higher bids from users and so on, which may indicate the occurrence of an interruption. Users monitor the price changes and a backup will be made when a rising edge of price changes occurs. The third benchmark algorithm, Ada-Rising-Edge-Driven, is an adaptive backup strategy based on Rising-Edge-Driven and proposed by Yi et al. in [27]. All the three benchmark algorithms as well as our Online-Spot algorithm and Offline optimal algorithm, are carried out for each historical price trace of the AWS spot instances and all the incurred costs are normalized to Hour-Boundary.

2) COST PERFORMANCE EVALUATION

We plot the average ratio of cost savings of all algorithms in Fig. 16 and CDF in Figs. 12, 13, 14 and 15. Let’s look at Fig. 12, which shows a stable market environment. Rising-Edge-Driven and Ada-Rising-Edge-Driven have performance close to Offline optimal algorithm. Ada-Rising-Edge-Driven, because of its adaptivity, is better than Rising-Edge-Driven in most cases. However, when the price fluctuates dramatically as shown in Fig. 14, rising edge occurs too frequently so that too many inappropriate backups will be made, leading to a sharp increase in extra backup costs. Overall, these two algorithms can save no more than 30% cost compared with the normalized algorithm. Online-Spot is better than the two algorithms and achieves 70% cost savings, compared with backing up every hour. As we can observe from Fig. 13, backing up every hour
has become a nice backup option in this data set which shows the most frequent price changes. Even Offline optimal algorithm can only save 30% cost more than Hour-Boundary. Rising-Edge-Driven and Ada-Rising-Edge-Driven perform even worse than Hour-Boundary. However, Online-Spot can still guarantee a stable competitive ratio compared with the Offline algorithm. There is more than 80% of the data cutting their costs while running the Online-Spot algorithm, compared with running Hour-Boundary algorithm. In general, the change of instance price depends on many aspects. It is impossible for the rising edge of price changes to predict the arrival of interruptions correctly all the time. Our algorithm formally evaluates the risk of interruption at every moment and makes a backup decision online, which can significantly reduce the cost in the case of medium and high interruption rates.

VI. CONCLUSION AND FUTURE WORK

Using spot instance can greatly reduce costs but carrying the risk of being interrupted. Cloud providers advise users to make backups when using spot instances. However, an arbitrary backup decision will incur unnecessary extra cost. In this work, we propose an online algorithm to help users to determine when to back up data when using spot instances, which can help users to achieve great cost savings while without requiring any a priori knowledge of future interruptions. We prove that the competitive ratio between our online algorithm and optimal offline algorithm is less than 2. Through the historical price of more than two thousands AWS spot instances for simulations, we show that our online algorithm can save cost significantly especially when interruptions occur frequently.

One situation we haven’t dealt with is when a user chooses multiple types of spot instances. When some instances are interrupted, other instances can take over the work of these instances to optimize time and cost. Our future work is to develop new online algorithms to deal with the above more complex scenarios.

REFERENCES

[1] Data Processing and Hosting Services Market—Growth, Trends and Forecast (2020-2025). Accessed: Apr. 5, 2020. [Online]. Available: https://www.researchandmarkets.com/reports/48697134/data-processing-and-hosting-services-market

[2] R. Lovas, E. Nagy, and J. Kovács, “Cloud agnostic big data platform focusing on scalability and cost-efficiency,” Adv. Eng. Softw., vol. 125, pp. 167–177, Nov. 2018.

[3] B. Cui, P. Shi, W. Qi, and M. Li, “Uploading multiply deferrable big data to the cloud platform using cost-effective online algorithms,” Future Gener. Comput. Syst., vol. 67, pp. 276–285, Feb. 2017.

[4] M. T. Islam, S. N. Siramana, S. Karunasekera, and R. Buyya, “Cost-efficient dynamic scheduling of big data applications in Apache spark on cloud,” J. Syst. Softw., vol. 162, Apr. 2020, Art. no. 110515.

[5] AWS EC2 Instance Price. Accessed: Mar. 11, 2020. [Online]. Available: https://aws.amazon.com/ec2/spot/

[6] AWS Case Studies. Accessed: Mar. 11, 2020. [Online]. Available: http://aws.amazon.com/solutions/case-studies/

[7] Y. Yao, Y. Gao, Y. Chen, Z. Guo, B. Chen, and T. Moscibrodz, “TR-spark: Transparent computing for big data analytics,” in Proc. 7th ACM Symp. Cloud Comput., Oct. 2016, pp. 484–496.

[8] S. Tang, J. Yuan, and X.-Y. Li, “Towards optimal bidding strategy for amazon EC2 cloud spot instance,” in Proc. IEEE 5th Int. Conf. Cloud Comput., Jun. 2012, pp. 91–98.

[9] M. Khodak, L. Cheng, A. S. Lan, C. Joe-Wong, and M. Chiang, “Learning cloud dynamics to optimize spot instance bidding strategies,” in Proc. IEEE INFOCOM-IEEE Comput. Commun., Apr. 2018, pp. 2762–2770.

[10] S. Tang, J. Yuan, C. Wang, and X.-Y. Li, “A framework for amazon EC2 bidding strategy under SLA constraints,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 1, pp. 2–11, Jan. 2014.

[11] D. Poola, K. Ramamohanarao, and R. Buyya, “Fault-tolerant workflow scheduling using spot instances on clouds,” Procedia Comput. Sci., vol. 29, pp. 523–533, Dec. 2014.

[12] Z. Xu, C. Stewart, N. Deng, and X. Wang, “Blending on-demand and spot instances to lower costs for in-memory storage,” in Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM), Apr. 2016, pp. 1–9.

[13] AWS SI Interrupt Mechanism. Accessed: Mar. 19, 2020. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html

[14] Amazon Simple Storage Service (Amazon S3). Accessed: Mar. 19, 2020. [Online]. Available: https://aws.amazon.com/ec2/3/

[15] Google Cloud Storage. Accessed: Mar. 19, 2020. [Online]. Available: https://cloud.google.com/storage

[16] S. Chaisiri, B.-S. Lee, and D. Niyato, “Optimization of resource provisioning cost in cloud computing,” IEEE Trans. Services Comput., vol. 5, no. 2, pp. 164–177, Apr. 2012.

[17] L. Chiariavligio, F. D’Andreaiovanni, R. Lancellotti, M. Shojafar, N. Sfeir-Melazi, and C. Canali, “An approach to balance maintenance costs and electricity consumption in cloud data centers,” IEEE Trans. Sustain. Comput., vol. 3, no. 4, pp. 274–288, Oct. 2018.

[18] Z. Wu, Z. Ni, L. Gu, and X. Liu, “A revised discrete particle swarm optimization for cloud workflow scheduling,” in Proc. IEEE 8th USENIX Conf. Netw. Syst. Design Implement., Jun. 2011, pp. 184–188.

[19] T. Wood, E. Czeczet, K. K. Ramakrishnan, P. Shenoy, and J. Merwe, “Disaster recovery as a cloud service: Economic benefits & deployment challenges,” in Proc. Conf. Hot Topics Cloud Comput. (HotCloud), 2010, pp. 8–14.

[20] W. Eric, F. Alireza, and I. Kannan, “Cross-cloud computing for capacity management and disaster recovery,” U.S. Patent 8 719 627, May 6, 2014.

[21] H. S. Gunawi, T. Do, P. Joshi, P. Alvaro, and J. M. Hellerstein, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, K. Sen, and D. Borthakur, “FATE and DESTINI: A framework for cloud recovery testing,” in Proc. 8th USENIX Conf. Netw. Syst. Design Implement., 2011, pp. 238–252.

[22] R. Singh, P. Sharma, D. Irwin, P. Shenoy, and K. K. Ramakrishnan, “Here today, gone tomorrow: Exploiting transient servers in datacenters,” IEEE Internet Comput., vol. 18, no. 4, pp. 22–29, Jul. 2014.

[23] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, “Deconstructing amazon EC2 spot instance pricing,” in Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci., Nov. 2011, pp. 304–311.

[24] J. Fabra, J. Ezpeleta, and P. Álvarez, “Reducing the price of resource provisioning using EC2 spot instances with prediction models,” Future Gener. Comput. Syst., vol. 96, pp. 348–367, Jul. 2019.
LIDUO LIN is currently pursuing the bachelor’s degree in software engineering with the School of Software, Shandong University, Jinan, China. His research interests include cloud computing and service computing.

LI PAN (Member, IEEE) received the Ph.D. degree from the Research Center of Human-Computer Interaction Virtual Reality, Shandong University, in 2011. She is currently an Associate Professor with the School of Software, Shandong University, and a member of the HCIVR Group. Her main research interests include service computing and cloud computing.

SHIJUN LIU (Senior Member, IEEE) received the B.S. degree in oceanography from the Ocean University of China and the M.S. and Ph.D. degrees in computer science from Shandong University, China. He is currently a Professor with Shandong University. His current research interests include services computing, enterprise services computing, and services system for manufacturing.

---

[25] W. Guo, K. Chen, Y. Wu, and W. Zheng, “Bidding for highly available services with low price in spot instance market,” in Proc. 24th Int. Symp. High-Perform. Parallel Distrib. Comput. (HPDC), 2015, pp. 191–202.

[26] I. Jangjaimon and N.-F. Tzeng, “Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing,” IEEE Trans. Comput., vol. 64, no. 2, pp. 396–409, Feb. 2015.

[27] S. Yi, D. Kondo, and A. Andrzejak, “Reducing costs of spot instances via checkpointing in the Amazon elastic compute cloud,” in Proc. IEEE 3rd Int. Conf. Cloud Comput., Jul. 2010, pp. 236–243.

[28] S. Yi, A. Andrzejak, and D. Kondo, “Monetary cost-aware checkpointing and migration on Amazon cloud spot instances,” IEEE Trans. Services Comput., vol. 5, no. 4, pp. 512–524, 4th Quart., 2012.

[29] A. Borodin and R. El-Yaniv, Online Computation and Competitive Analyses. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[30] R. Fleischer, “On the bahncard problem,” Theor. Comput. Sci., vol. 268, no. 1, pp. 161–174, Oct. 2001.

[31] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, “Competitive randomized algorithms for nonuniform problems,” Algorithmica, vol. 11, no. 6, pp. 542–571, Jun. 1994.

[32] A. Karlin, C. Kenyon, and D. Randall, “Dynamic TCP acknowledgment and other stories about e/(e–1),” Algorithmica, vol. 36, no. 3, pp. 209–224, Jul. 2003.

[33] J. Meyer, D. D. Gremler, and J. Hogreve, “Do service guarantees guarantee greater market value?” J. Service Res., vol. 17, no. 2, pp. 150–162, 2014.

[34] T. Fang, W. Mackillop, W. Jiang, A. Hildesheim, S. Wacholder, and B. E. Chen, “A Bayesian method for risk window estimation with application to HPV vaccine trial,” Comput. Statist. Data Anal., vol. 112, pp. 53–62, Aug. 2017.

[35] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[36] A. Borodin and R. El-Yaniv, Online Computation and Competitive Analysis. Cambridge, U.K.: Cambridge Univ. Press, 1998.