# Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances

Sangho Yi, *Member, IEEE,* Artur Andrzejak, and Derrick Kondo, *Member, IEEE*

**Abstract**—Recently introduced spot instances in the Amazon Elastic Compute Cloud (EC2) offer low resource costs in exchange for reduced reliability; these instances can be revoked abruptly due to price and demand fluctuations. Mechanisms and tools that deal with the cost-reliability trade-offs under this schema are of great value for users seeking to lessen their costs while maintaining high reliability. We study how mechanisms, namely, checkpointing and migration, can be used to minimize the cost and volatility of resource provisioning. Based on the real price history of EC2 spot instances, we compare several adaptive checkpointing schemes in terms of monetary costs and improvement of job completion times. We evaluate schemes that apply predictive methods for spot prices. Furthermore, we also study how work migration can improve task completion in the midst of failures while maintaining low monetary costs. Trace-based simulations show that our schemes can reduce significantly both monetary costs and task completion times of computation on spot instance.

**Index Terms**—Checkpointing, Reliability, Fault-tolerance, Cloud computing, Volatile resources.

## 1 Introduction

The vision of computing as a utility has reached new heights with the recent advent of Cloud Computing. Compute and storage resources can be allocated and deallocated almost instantaneously and transparently on an as-need basis.

Pricing of these resources also resembles a utility, and resource prices can differ in at least two ways. First prices can differ by vendor. The growing number of Cloud Computing vendors has created a diverse market with different pricing models for cost-cutting, resource-hungry users.

Second, prices can differ dynamically (as frequently as an hourly basis) based on current demand and supply in market-based resource management system. These systems have recently become widespread with data center providers, such Google Inc. [1], Amazon Inc. [2] and Enomaly [3]. By lowering server prices dynamically during periods of low demand, data center operators can increase the server utilization and overall profitability.

For instance, in December 2009, Amazon released spot instances, selling the spare capacity of their data centers. Their dynamic pricing model is based on bids by users. If the users' bid price is above the current spot instance price, their resource request is allocated. If at any time the current price is above the bid price, the request is terminated.

Spot instances are particularly interesting for compute-intensive jobs that are divisible. Divisible workloads, such video encoding and biological sequence search (BLAST, for example), are an important class of applications [4]. We believe this is a common type of application that is amenable to failure-prone spot instances. Compared to "classical" on-demand resources offered by EC2, spot instances have the benefit of a price discount surpassing 50% [5]. This monetary incentive can frequently offset the drawbacks of drastically changed availability behavior. Clearly, there is a trade-off between the cost of the instance and its reliability.

Due to relative novelty of this resource class, there are currently hardly any methods or middleware which could cope or leverage changes in pricing or reliability. While some task scheduling frameworks such as BOINC [6] are designed for managing highly volatile resources, none of them is capable of considering monetary costs. Ideally, the middleware would have mechanisms to seek by itself the cheapest source of computing power given the demands of the application and current pricing.

In this paper, we investigate two mechanisms, namely, checkpointing and work migration, which can be used to achieve the goal of minimizing monetary costs while maximizing reliability. Using real price traces of Amazon's spot instances, we study various dynamic checkpointing strategies that can adapt to the current instance price and show their benefit compared to static, cost-ignorant strategies. Our key result is that the dynamic checkpointing and work migration strategies significantly reduce the monetary cost and shorten job completion times.

The remainder of this paper is organized as follows. Section 2 introduces the spot instances in the Amazon Elastic Compute Cloud (EC2), characterizes failure distribution and recovery time for out-of-bid situations and describes checkpointing and work migration schemas. Section 3 evaluates performance of their combinations

- *S. Yi and D. Kondo are with INRIA Grenoble - Rhône Alpes, France. E-mail: sangho.yi@inria.fr, derrick.kondo@inria.fr*
- *A. Andrzejak is with Heidelberg University, Germany. E-mail: artur@uni-hd.de*

*This paper is an extended version of our previous work entitled "Reducing Costs of Spot Instances via Checkpointing in the Amazon's Elastic Compute Cloud", which was presented in IEEE Cloud 2010 conference.*

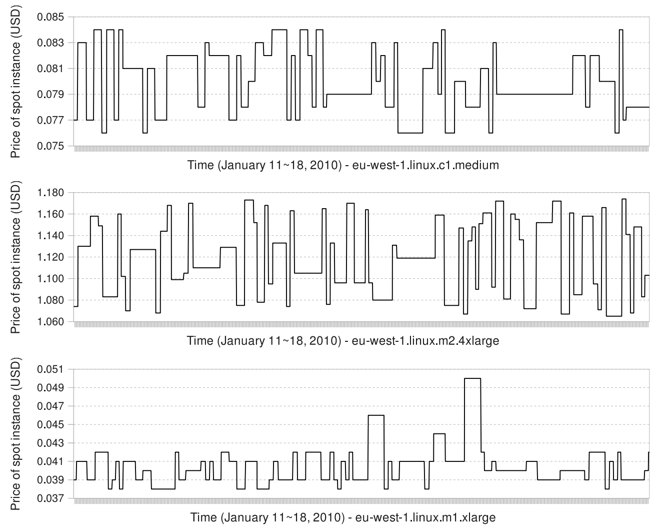

Fig. 1. Spot price fluctuations of *eu-west-1.linux* instance types

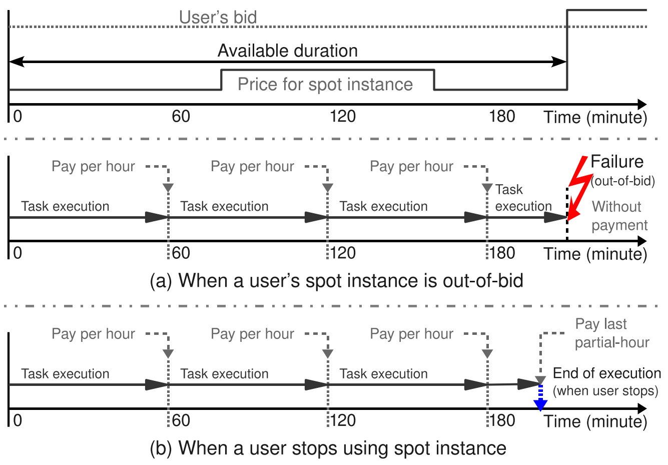

Fig. 2. Examples of pricing for Amazon's spot instance

based on the previous price history of the spot instances. Section 4 describes related work. Finally, Section 5 presents conclusions and possible extensions of this work.

## 2 Spot Instances on Amazon EC2

### 2.1 System model

Amazon allows users to bid on unused EC2 capacity provided as 64 types of *spot instances* that differ by computing / memory capacity, OS type and geographical location [2]. Their prices called *spot prices* change dynamically based on supply and demand. Figure 1 shows examples of spot price fluctuations for three *eu-west-1.linux* instance types during 8 days in January 2010. Customers whose bids meet or exceed the current spot price gain access to the requested resources. Figure 2 shows how Amazon EC2 charges *per-hour* price for using a spot instance. The following system model was made according to the characteristics of Amazon EC2's spot instances:

◇ Amazon provides a spot instance when a user's bid is greater than the current price.
◇ Amazon stops immediately without any notice when a user's bid is less than or equal to the current price. We call this an *out-of-bid* event or a *failure*.
◇ Amazon does not charge the latest partial hour when Amazon stops an instance.
◇ Amazon charges the last partial hour when a user terminates an instance.
◇ The price of a *partial-hour* is considered the same as a *full-hour*.
◇ Amazon charges each hour by the last price.
◇ Amazon freely provides the spot price history.
◇ The price of Amazon's S3 (simple storage service) is negligible[1].

### 2.2 Overview of the approaches

We assume a user is submitting compute-intensive jobs that are divisible. Divisible workloads, such video encoding and biological sequence search (BLAST, for example), are an important class of application [4]. We believe this is a common type of application that is amenable to failure-prone spot instances. By setting the *bid price* a user can partially control the monetary cost of job execution in this scenario. However, a lower bid price implies more frequent failures which in turn leads to two detrimental effects:

◇ part of the computation is lost and must be repeated, thus increasing the monetary cost
◇ the overall job execution time increases.

In this work we study two mechanisms for reducing these negative effects: checkpointing and work migration.

**Checkpointing.** This traditional technique stores persistently snapshots of the current application state and uses them for restarting the execution at a later time. The critical issue here is at which times and whether checkpoints are taken. While abundant checkpoints lead to high cumulative overhead (as each checkpoint means lost computation time), infrequent or wrongly timed checkpoints increase the recovery time after a failure and cause repeated computation. The focus of this work are policies for *deciding* when and whether a checkpoint should be taken (see Section 2.3) and evaluating them in regard to minimizing the induced monetary costs.

In case of an out-of-bid event all spot instances with below-market bid price are terminated. Consequently, user bids again to re-acquire resources, and when those

1. Amazon provides 1 GB-month storage service with the price of 0.15 USD. This is much lower than the price of computation during one month [7].

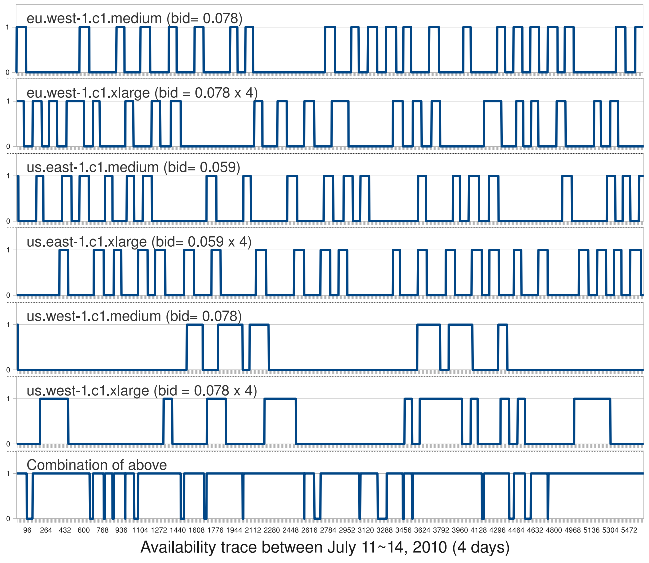

Fig. 3. Combined availability of comparable instance types for work migration (1: available, 0: unavailable)

are granted, she restarts the execution (Fig. 7). This involves loading of the last saved application state from persistent storage. We assume here that Amazon's S3 service is used. As noted above, the monetary cost of this storage is insignificant compared to lost (yet paid) computation time. We thus consider only the latter as the cost of checkpointing and recovery. As the actual implementation of checkpointing and recovery is application-specific, it is not considered in this study.

**Work migration.** The above schema allows to change the *instance type* after a failure provided that application is flexible enough to handle different number of cores; however, in the scenario of divisible workloads this capability can be assumed. The idea is to bid for another instance type (than recently used one) if this new instance type can be acquired at a *per-core* price comparable to user's last bid price. In this way the waiting time until recovery will be almost eliminated. In [5], [8], [9] we have observed that low bid prices cause job completion time to increase exponentially. By eliminating the waiting time until re-acquiring resources this technique effectively reduces the job completion time without significant cost increase.

Figure 3 illustrates the benefits of this approach: at fixed bid level, the combined availability of comparable spot instance types (high-CPU instances in EU and US) is much higher than that of a single instance type. In this example, we plotted 7 days of availability (as $1$) and unavailability (as $0$), and assumed a low bid price. If we utilize $6$ instance types, the combined availability is almost $92\%$ during $7$ days, while one instance type is available only $15{\sim}25\%$ during this period (at a comparable per-core bid price level).

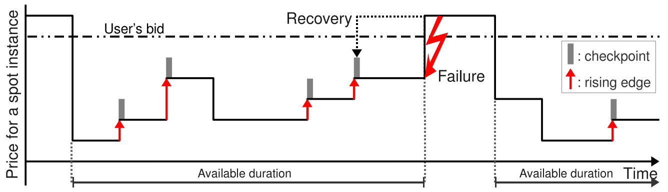

Fig. 4. Rising edge-driven checkpointing

## 2.3 Checkpointing schemes

We describe in this section the proposed checkpointing schemes. Table 1 gives an overview of them.

### 2.3.1 *Baseline comparison schemas - OPT and NONE*

Two methods without practical value are included in the evaluation for comparison. The *optimal* schema OPT takes a checkpoint just prior to a failure and represents the best possible decision strategy. The *no checkpointing* schema takes no checkpoints, and has to recompute everything since the beginning in case of a failure.

### 2.3.2 *Hourly checkpointing - HOUR*

In this schema checkpoints are taken periodically at the boundary of each hour since start of resource usage. It is the most intuitive one for the spot instances, because an *hour* is the lowest granularity of spot instance pricing. It also provides a guarantee of paying for the actual progress of computation.

### 2.3.3 *Rising edge-driven checkpointing - EDGE*

Here a checkpoint is taken at each increase of spot price of the currently used instance type, see Figure 4. Such price increase is termed here a *rising edge*. Each such event increases the probability of an out-of-bid situation, as the difference between spot price and current bid shrinks.

In the previous work [8], we showed that "edge-driven" checkpointing was suboptimal because of the bursty fluctuations of the spot price curves. This leads to abundant checkpointing and thus high overhead. We include this heuristic here for completeness and for comparison.

### 2.3.4 *Basic adaptive checkpointing - A*

In this approach we decide every $10$ minutes whether to take or to skip a checkpoint. The decision is based on the expected recovery time $R(t)$ in case of a (future) failure. This time is computed for both cases: if a checkpoint is taken ($R_{take}$) or if it is skipped ($R_{skip}$). Clearly, this schema takes a checkpoint if $R_{skip} > R_{take}$ and skips it otherwise.

This decision significantly affects the recovery time if a failure occurs, and thus the execution time of the running task. Section 2.5 describes how both expected times are computed. The major input of this computation is the probability of a failure depending on the original bid price, time since start of the availability interval and time since last checkpoint.

#### 2.3.5 Current-price based adaptive checkpointing - C

This schema is closely related to the previous one and decides every 10 minutes whether to take or to skip a checkpoint. However, decision is based on more current information for computing the probability of a failure. In addition to the factors influencing this probability mentioned above (Sec. 2.3.4), we also consider the spot price at the time of taking the decision. The rationale here is that if current spot price is close to the original bid price, the latter is more likely to be surpassed by the spot price, leading to an out-of-bid event. The details are presented in Section 2.5.

### 2.4 Work migration

Independently of the checkpointing schema we can decide to change the instance type upon a failure as outlined in Section 2.2. Heuristics for work migration must decide on two factors. First, they must decide on how to migrate, that is, what bid price to use on the new instance. Second, they must decide on which instance type to migrate to.

With respect to how to migrate, our heuristics will place an identical bid for the new instance type. With respect to which instance type to migrate to, our heuristics are described as follows:

LP **Lowest price.** The heuristic chooses the instance type with the lowest current price. The intuition is that the lower the current price, the less likely out-of-bid events will occur.

LF **Lowest failure rate.** The heuristic chooses the instance type with the lowest failure rate. Failure distributions are computed dynamically for each instance type, given their price trace. The intuition is that the failure rate considers both the user bid and the price with respect to the likelihood of out-of-bid events.

HF **Highest failure rate.** The heuristic chooses the instance type with the highest failure rate. The intuition is that the higher the failure rates, the higher the potential to exploit uncharged partial-hours as described in Sec. 2.6.3.

Note that selecting an optimal instance on migration is impossible because of ignorance of the forthcoming price changing curves. However, there are several heuristics of selecting the next instance type. We can select based on the current price, or the failure arrival rate. For example, if we want to have less failures, we can use the “lowest-failure-arrival-first” policy, or if we want to minimize the monetary cost, the “lowest-price-first” policy is also possible. In this work we evaluate 3 different policies on selecting the next instance type.

### 2.5 Adaptive checkpointing

In this section we describe the methods for adaptive deciding on taking or skipping a checkpoint in schemas A (Sec. 2.3.4) and C (Sec. 2.3.5). The major factor influencing this decision is the expected recovery time, which is turn influenced by the probability of an out-of-bid (i.e. failure) event in the near future. Here we derive formulas to compute the expected recovery time under any probability distribution and discuss the characteristics of failure probability distributions observed in EC2 spot instance data.

TABLE 1
Checkpointing schemas and their options

| Schema | Description |
|---|---|
| OPT | optimal strategy |
| NONE | no checkpointing |
| HOUR | hourly checkpointing |
| EDGE | rising edge-driven checkpointing |
| A | basic adaptive checkpointing |
| C | current-price based adaptive checkpointing |
| **Option** | **Description** |
| +$M$ | deploy work migration after a failure:<br>$M_{LP}$: select the “lowest price” instance for migration<br>$M_{LF}$: select the “lowest failure rate” instance<br>$M_{HF}$: select the “highest failure rate” instance |
| +$(T_w)$ | use $T_w$ days of past traces to estimate probability density function of failure occurrences |

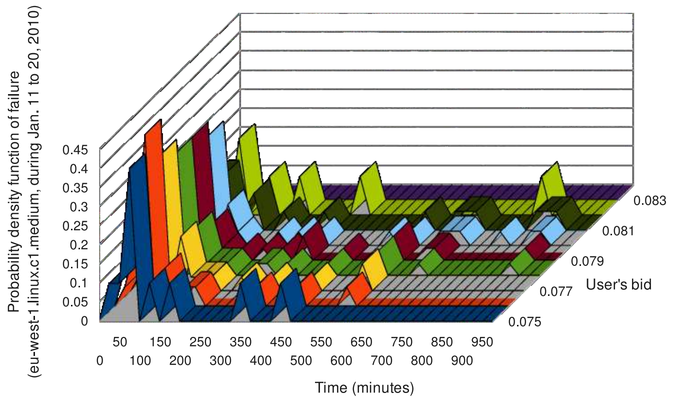

Fig. 5. Examples of probability density function of failure (out-of-bid) occurrence $f(t, p_b)$ on *eu-west-1.linux-c1.medium* instance type

#### 2.5.1 Failure probability distributions

We are interested in estimating the probability of an instance failure (i.e. out-of-bid situation) within a time interval $t$ (usually it is a future time interval starting at the ”current” time $t_0$ at which a checkpointing decision is taken). It is quite plausible to assume that this probability depends on the bid price $p_b$ at which the instance has been acquired. Therefore, we model this probability by a discrete *probability density function* $f(t, p_b)$ of failure occurrences, where $p_b$ is a parameter and $t$ the variable of the distribution. Figure 5 shows that (for the instance type *eu-west-1.linux-c1.medium*) $f(t, p_b)$ indeed depends on both the time interval and the user’s bid. Note that for the low bid price of $0.075$ we have high probability that an out-of-bid event occurs fast (already within the

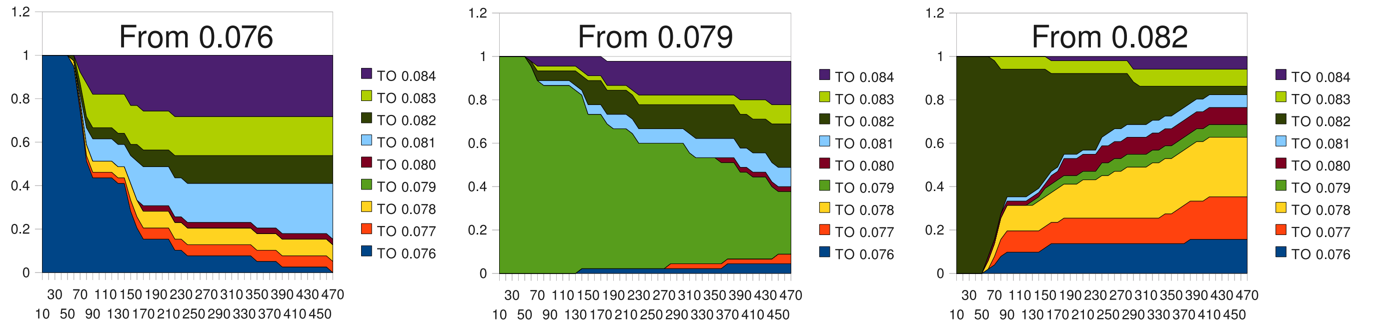

Fig. 6. Probability distributions of prices depending on the starting price (”From”) and time on *eu-west-1.linux-c1.medium* instance type (x-axis: time in minutes, y-axis: probability)

first 100 minutes). However, for high bid price of 0.081 the out-of-bid event is less likely in the same time interval as this the spot price rarely surpasses 0.081. Obviously $f(t, p_b)$ can be approximated via a histogram of availability durations (for each bid price $p_b$) created from the history of price fluctuations.

**Considering current price.** When estimating $f(t, p_b)$ at a certain time $t_0$ we can also take into account the *current* spot price $p_c$ (i.e. at time $t_0$) as an additional information. Therefore we introduce a refined probability density function $f_p(t, p_b, p_c)$ of failure occurrences where $p_c$ and $p_b$ are parameters and $t$ is again the distribution variable.

Figure 6 illustrates that $p_c$ contains essential information to estimate failure probability (here the same instance type as in Figure 5 is used). It shows the transition probabilities from the spot price at the begin of an interval (”From:” label) to a new spot price (”To:” prices in legend) depending on the interval duration (shown on the $x$-axis). A line parallel to $y$-axis intersecting $x$-axis at interval duration $t$ (in minutes) gives for each possible price the probability (expressed as line segment length) that it is assumed within time $t$.

For example, starting at $0.076$ spot price, we have probability of about 0.3 that spot price of 0.084 has been assumed within 270 minutes and the probability of about 0.6 that the new spot price is 0.082, 0.083 or 0.084. Consequently, we have a failure probability 0.6 for a bid price $p_b$ of 0.081 or below in this time interval. However, for the same bid price but initial spot price of $0.079$ (middle chart in Figure 6) we have a probability of only 0.35 for a failure within the same time duration. This effect can be explained by the fact that the lower-end spot price of 0.076 is less stable and prone to ”jumps” even to high levels than the mid-range spot price of 0.079. The comparison shows that the starting price gives an essential information for estimating the failure probability distribution. Contrary to this, the simple pdf $f(t, p_b)$ (Figure 5) averages over *all possible* spot prices at the beginning of the time interval.

**Memoryless and memoryfull assumptions.** Obviously the future price is dependent on the current spot price. Thus, the distribution of the time to failure depends on the current price. Because of this dependence, the failure distribution likely cannot be modeled with an exponential distribution, which has a memoryless property. If the failure distribution is memoryless, the probability of a failure after time $t$ is the same as the probability of a failure after time $s + t$, given no failure during time $s$. Consequently, the failure distribution is not memoryless and cannot be modeled by exponential distribution. In previous work [8], we assumed that the availability distribution follows exponential distribution, which means the failure arrival rate $f(t, p_b)$ has a memoryless property. In this work we correct this assumption and modify the formulas in Section 2.5.2 to work with *any* failure distribution.

TABLE 2
Notations and symbols for adaptive checkpointing

| Notation | Description |
|---|---|
| $p_b$ | bid price on a spot instance type |
| $p_c$ | current price on a spot instance type (at time $t_p$) |
| $t_p$ | present time (i.e. time of deciding on skipping or taking a checkpoint) |
| $r$ | time to restart a task |
| $t_r$ | number of time units needed to complete the job |
| $t_c$ | time to take a checkpoint |
| $f(t, p_b)$ | probability density function of a failure within time interval $[t_p, t_p + t]$ and with bid at price $p_b$ |
| $f_p(t, p_b, p_c)$ | probability density function of a failure within time interval $[t_p, t_p + t]$ with bid at price $p_b$ and considering the spot price $p_c$ at time $t_p$ |
| $T(w, t_p)$ | expected execution time of a job with work requirement of $w$ time units without checkpointing when starting at time $t_p$ |
| $R_{take}(t, t_p)$ | expected recovery time if a checkpoint at time $t_p$ is taken and previous checkpoint was at time $t_p - t$ |
| $R_{skip}(t, t_p)$ | expected recovery time if a checkpoint at time $t_p$ is skipped and previous checkpoint at time $t_p - t$ |

### 2.5.2 Expected recovery time

Adaptive schemas A (Sec. 2.3.4) and C (Sec. 2.3.5) require knowledge of the expected recovery time in case of a failure. In this section we generalize and modify results from [10] to derive formulas for these times. Table 2 describes notations and symbols relevant for adaptive checkpointing.

As the expected recovery times are dependent on the current spot price (since failure distributions depend on

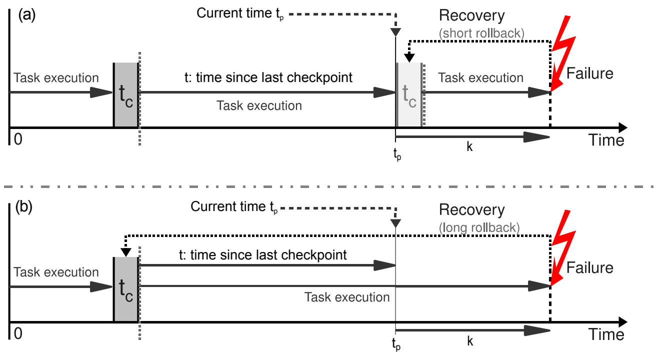

Fig. 7. Effects of taking (a) and skipping (b) a checkpoint on the expected recovery time

them, see Section 2.5.1) we introduce as parameter the time $t_p$ (”present time”) at which the current price $p_c$ is observed (i.e. a decision about the current checkpoint is taken). Let $t$ be time interval (in units) between last checkpoint and $t_p$, see Figure 7. Then $R_{take}(t, t_p)$ is the expected recovery time if a checkpoint at time $t_p$ is *taken*. Analogously, $R_{skip}(t, t_p)$ is the expected recovery time if a checkpoint at time $t_p$ is *skipped*.

To evaluate formulas for expected recovery times we need the following theorem, where $r$ is the overhead time to restart a task after a failure. In the following, $f(t)$ describes a probability density function of a failure (either $f(t, p_b)$ or $f_p(t, p_b, p_c)$).

*Theorem 1:* Let $w$ be number of time units needed to execute a (partial) job without failures. Then the expected execution time $T(w, t_p)$ for this work in presence of failures, without checkpointing and when starting at time $t_p$ is

$$\frac{w\sum_{k=w}^{\infty} f(k+t_p) + \sum_{k=0}^{w-1}(k+r)f(k+t_p)}{1-\sum_{k=0}^{w-1} f(k+t_p)}. \qquad (1)$$

*Proof:* Let $k$ be the index of the time unit (relative to $t_p$) in which the first out-of-bid event occurs, see Figure 7. The conditional expected execution time is then:

$$T(w,t_p) = \begin{cases} w & \text{if } k{\geq}w \\ k+r+T(w,t_p) & \text{otherwise.} \end{cases}$$

By the law of total expectation

$$\begin{aligned} T(w,t_p) = & w\sum_{k=w}^{\infty} f(k+t_p) \\ & + \sum_{k=0}^{w-1}(k+r+T(w,t_p))f(k+t_p). \end{aligned}$$

Rearranging with respect to $T(w, t_p)$ we obtain the desired result. □

In the following, let $t_r$ be the number of time units needed to complete the job (relative to time $t_p$).

*Theorem 2:* Let $t$ be number of time units since the last checkpoint and $t_p$ the current time. Then the expected recovery time $R_{skip}(t, t_p)$ when skipping a checkpoint at time $t_p$ is given by

$$R_{skip}(t,t_p) = \sum_{k=0}^{t_r-1}(k+r+T(t,t_p))f(k+t_p). \qquad (2)$$

*Proof:* When a failure occurs at time unit $k$ and within $t_r$ time units needed to complete the job, the task should be re-executed from the last checkpoint (taken at the time $t_p - t$). This implies recomputing work $t + k$ (in time units). Thus,

$$R_{skip}(t,t_p) = \begin{cases} k+r+T(t,t_p) & \text{if } k < t_r \\ 0 & \text{otherwise} \end{cases}$$

By integrating above, we obtain Eq. (2). □

Let $t_c$ be the time to take a checkpoint, see Table 2.

*Theorem 3:* Let $t$ be number of time units since the last checkpoint and $t_p$ the current time. Then the expected recovery time $R_{take}(t, t_p)$ when taking a checkpoint at time $t_p$ is given by

$$\begin{aligned} R_{take}(t,t_p) = & \sum_{k=0}^{t_r-1}(k+r)f(k+t_p) + t_c\sum_{k=t_r}^{\infty} f(k+t_p) \\ & + T(t,t_p-t)\sum_{k=0}^{t_c-1} f(k+t_p). \end{aligned} \qquad (3)$$

*Proof:* When a failure occurs at time unit $k$ (relative to $t_p$) within $t_c$ time units (i.e. $k \leq t_c$), the task should be re-executed from the last checkpoint, and when a failure occurs in $t_c \leq k < t_r$ the task can be recovered from the new checkpoint. In addition, $R_{take}(t, t_p)$ has overhead $T(t_c, t_p)$ of taking a checkpoint, and thus,

$$R_{take}(t,t_p) = \begin{cases} k+r+T(t,t_p) & \text{if } k < t_c \\ k+r & \text{else if } t_c \leq k < t_r \\ t_c & \text{otherwise} \end{cases}$$

By the law of total expectation, and simplifying it with $k + r$, we obtain Eq. (3). □

### 2.5.3 Variations of the adaptive checkpointing

In case of the checkpointing schema A (Sec. 2.3.4) we use as $f(t)$ in Section 2.5.2 the ”simple” failure probability $f(t, p_b)$. In this case the parameter $t_p$ is not essential since $f(t, p_b)$ does not depend on the price at the start of the considered time interval. For more sophisticated schema C (Sec. 2.3.5) $f_p(t, p_b, p_c)$ is used for $f(t)$. Thus, to evaluate the formulas in Section 2.5.2 we consider a separate pdf of failure probabilities for each different current spot price (depending on the time $t_p$).

As noted above, both $f(t, p_b)$ and $f_p(t, p_b, p_c)$ are estimated from the price traces of EC2 spot instances. Given that the characteristics of the spot prices change over time there is trade-off how much of the historical traces (relative to time of the analysis) should be used. If we use too little of past traces, we may lose long-term trends. Otherwise, we may lose recent behaviors of price fluctuations. To evaluate this effect we introduce the *window size* $T_w$ of the past traces. The optimal value of this parameter should maximize the benefit of using past traces to expect forthcoming failure arrivals.

## 2.6 Further aspects

### 2.6.1 Generalizing the approaches

Approaches presented in previous sections have been introduced in context of Amazon EC2 spot instances model, but in fact they generalize easily to other system models where resources can be aquired at time-fluctuating prices. To facilitate a possible transfer, we make here our assumptions explicit.

Thus, the checkpointing schemas listed in Section 2.3 two requirements must be fulfilled: (i) the cost of external storage used for storing application state snapshots is negligible and (ii) the price history of resources is available. Except for these assumptions, we exploit the hourly checkpointing schema (HOUR) is motivated by the fact that EC2 spot prices change in only hourly intervals. However, even for this special schema any other period of price updates would work. Furthermore, our assumption of ”memoryfull” price and failure distributions (Section 2.5.1) covers a very general case, especially it includes memoryless cases. Note that we consider in Section 2.6.3 strategies which are very specific for the EC2 spot instance model yet these are independent of the above-presented schemas.

As for the work migration schemas, our implicit assumptions are that (i) there exist multiple instance types with comparable per-core prices and (ii) the effort of changing the instance type upon a failure is negligible. These assumptions might not be valid for other providers than Amazon. While (i) is mandatory for the work migration to make sense, potential monetary or time cost incurred by change of the instance type can be easily considered when transferring this approach to other provider scenarios.

Summarizing, the *schemas* and *approaches* presented above are either directly applicable to other Cloud vendors or require minimal changes. On the other side, the evaluation results presented in Section 3 are strongly dependent on EC2 spot prices. Therefore, we include a comparative study for other Cloud vendors as a part of the future work (to be done as soon as resource prices of alternative vendors are available).

### 2.6.2 Combining checkpointing and work migration

The checkpointing schemes presented in Section 2.3 can be combined with the work migration schemas from Section 2.4 in an *orthogonal* way. Consequently, we obtain 20 different *checkpointing policies*: there are 5 different schemes (disregarding NONE), and 4 ways of work migration (low price, high failure, low failure, none migration). Furthermore, changing the window size $T_w$ of past traces further enlarges the number of the possible policies. Section 3 presents this evaluation of a number of relevant cases.

### 2.6.3 Exploiting Amazon EC2's spot pricing rules

According to Amazon EC2's spot pricing rules shown in Fig. 1 users do not pay the partial-hour when Amazon terminates the running instances. Thus, it is possible to exploit this fact if we have precise expectation of the forthcoming failures, and if we select the appropriate starting (or, recovering) points on each availability duration.

**Circumventing payment of the last hour.** Amazon may terminate the running task with certain probability $s_x = \sum_{k=0}^{k=x} f_p(k+t_p, p_b, p_c)$ where $x$ is the time remaining to the hour-boundary.

If a user delays termination of the running instance up to the hour-boundary, she might benefit from about expected $s_x \times p_c$ cost reduction for each different job request. This is a way to circumvent payment of the last hour, and users can utilize this approach for short-term tasks (or task whose running times are known with high accuracy).

Note that selecting an appropriate starting (recovering) points on each availability duration is helpful to maximize the probability $s_x$. However, in this work we neglect this effect and restart tasks as soon as we re-acquire the resources. In this way, the impact of the delayed termination does not depend on the considered checkpointing policy.

**Fluctuations make difference.** In [8] we found that bidding lower than mean price provides very frequent fluctuations, and thus, it has more chance to utilize the non-paying partial-hours. This is reason why the OPT schema is much better than that of the HOUR (hourly) checkpointing schema when the bid price is lower than mean. Using migration on failures can utilize such fluctuations to reduce monetary cost, because we can select an appropriate instance type which is expected to get failed within a hour. If the number of instance types and zones will be expanded in the future, exploiting the fluctuations via work migration will become even more interesting.

## 3 Evaluation of the Checkpointing Policies

In this section, we analyze the impact of checkpointing policies on 42 spot instance types in Amazon EC2[2]. We simulated the checkpointing policies based on the real price traces in terms of the job completion time, total monetary cost, and the product of *monetary cost×completion time* with several adjustable parameters.

## 3.1 Simulation Setup

Table 3 shows our simulation setup in detail. We assume that the checkpointing cost of running programs is known. We used the constant value for $t_c$, but using a variable checkpointing cost is also possible in our system

2. We did not use all 64 instance types on Amazon EC2, because some instances do not have a large enough amount of price traces.

TABLE 3
Values of parameters used in this paper

| Parameter | Value |
|---|---|
| Starting date of traces for failure prediction | Jan. 11th, 2010 |
| Ending date of traces for failure prediction | Jul. 10th, 2010 |
| Starting date of traces for evaluation | Jul. 11th, 2010 |
| Ending date of traces for evaluation | Sep. 29th, 2010 |
| Minimum bidding granularity | 0.001 USD |

| Parameter | $r$ | $t_c$ | $t_r$ |
|---|---|---|---|
| Value | 10 mins | 2 mins | 1,000 mins |

model. We assume that the total work of each job is $1,000$ minutes, and we used several different numbers of days of previous price history to get the probability density function of the availability durations.

We implemented a simulator that reads the past history of spot price, calculates the probability density function of availability durations and rising edges, and simulates $20+$ combinations of checkpointing policies (see Table 1) on the $42$ types of spot instances. For each data point, we simulated 100 experiments to ensure confidence of our results. In the following sections, we show only the most effective or interesting combinations of checkpoint policies (versus all combinations) for clarity.

## 3.2 Simulation Results and Evaluation

In the following, the policy OPT serves as a comparison baseline as it is optimal in the sense that checkpoints are taken directly before a failure. The policy HOUR serves as another (conservative) comparison baseline where users pay only for ”saved” computation results.

We have picked the *eu-west-1.linux.m1.large* as a representative instance type to evaluate the total monetary cost of a task, its completion time, and a product of both as a combined metric.

### 3.2.1 Impact of checkpointing schemas

First of all, we present the impact of 6 checkpointing schemas described in Table 1 with fixed parameters (with a window size of $T_w = 70$ days of past traces and without migration).

**Total price.** Figure 8 shows the total monetary cost for executing a given task ($1,000$ minutes) for the investigated instance type. Obviously the edge-driven checkpointing schema performs poorly which can be attributed to the overhead of too frequent checkpoints in this case. The adaptive checkpointing schemas (A and C) result in lower costs compared with the other possible checkpointing schemas. As expected, among these two the heuristic C that is aware of the current price shows better performance than the heuristic A that ignores the current price. However, we still have a $10 \sim 20$ percent difference between OPT and the realistic schemas.

**Task completion time.** Figure 9 shows the task completion time for the *eu-west-1.linux.m1.large* instance type. We can observe similar differences in efficiency among checkpointing schemas as for the monetary costs in

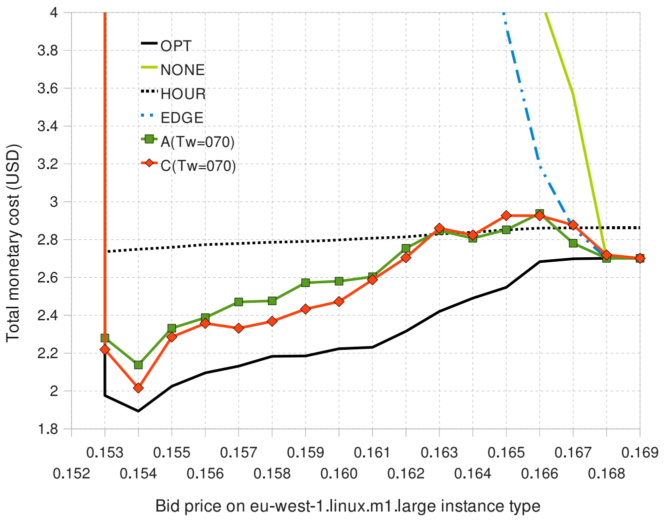

Fig. 8. Total monetary cost of task execution

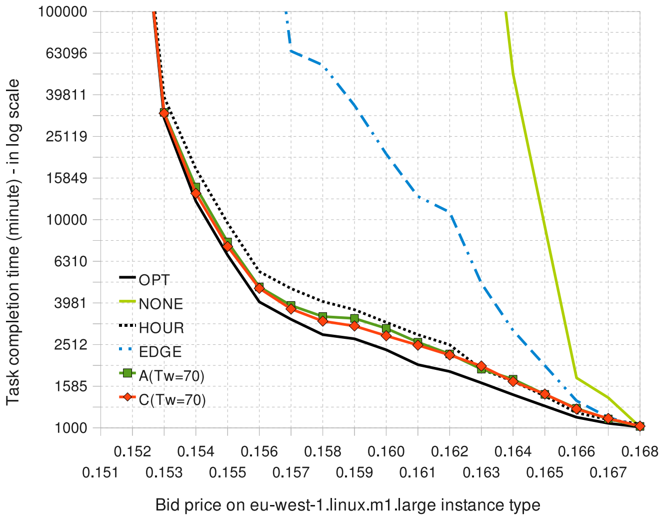

Fig. 9. Task completion time

Fig. 8. The most interesting fact is that a lower bid prices produce longer task completion times. This is because the fraction of time that an instance is available decreases with bid price. For instance, if a user has very low bid price, the instance is infrequently available, and the continuous periods of availability are relatively short. Still, the difference between OPT and the other schemas is about $10 \sim 15$ percent.

**Normalized combined metrics.** Figure 10 shows the combined performance metrics i.e., the product of the total monetary cost and task completion time. Furthermore, this product has been normalized by dividing through the same product for OPT. Obviously adaptive checkpointing aware of the current price (C) is better than the others for almost all bid values. We also observe that the performance gap between OPT and the best schema (C) is about $20 \sim 35$ percent.

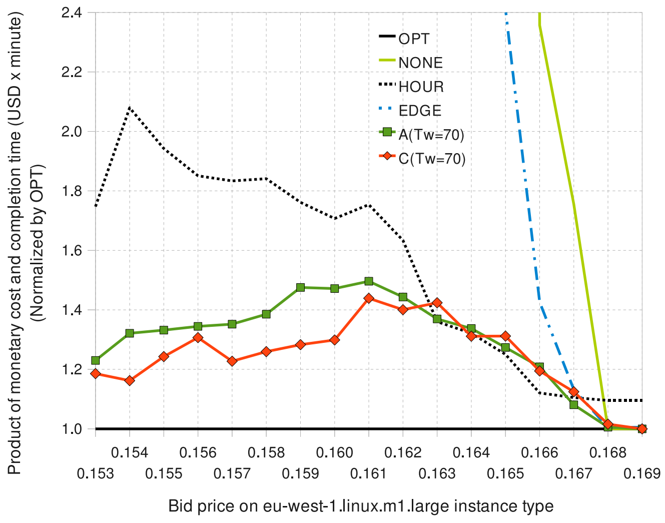

Fig. 10. Product of total price and task completion time

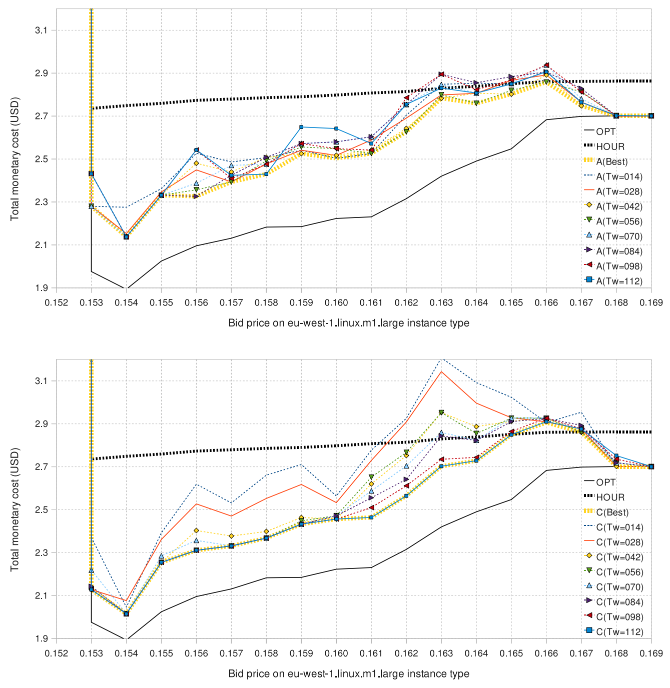

Fig. 11. Total monetary cost of adaptive checkpointing according to window sizes (upper figure: schema A, lower figure: schema C)

### 3.2.2 *Impact of the window size $T_w$ of past traces on schemas A and C*

We study the impact of the amount $T_w$ of past traces on adaptive checkpointing decision (see Section 2.5.3). Our testing targets are the two adaptive checkpointing schemas (A and C), which are working with probability distribution function of failures. In general, shorter window size is better if we need to catch recent behavior, while longer window size is better when we need to have stable statistical expectation. Thus, it depends on

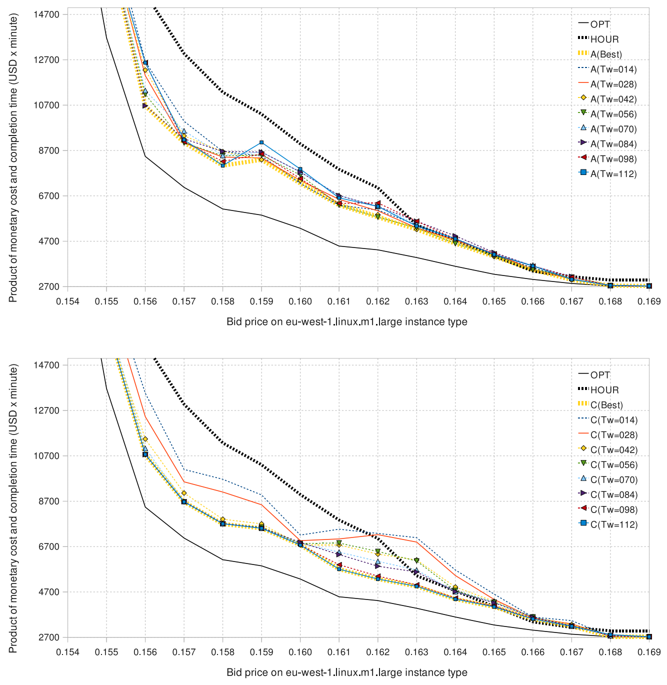

Fig. 12. Task completion time of adaptive checkpointing according to different window sizes (upper figure: schema A, lower figure: schema C)

the trends of users' demands on the spot market, or it may be affected by Amazon's supply and price regulation.

Figure 11 shows the total monetary cost of the adaptive checkpointing schemas with different window sizes. In case of basic adaptive checkpointing (A), it is difficult to determine the best window size for the given bid price. In some cases, a smaller window size is better than a longer one, while a longer window size is better on another bid prices. On the other hand, in the results of schema C (aware of current price), we can observe that a longer window size is better to reduce the total monetary cost in almost the entire bid range. In addition, C provides better performance than A when the bid price is smaller than $0.163$ USD, which is around mean price ($0.161$ USD). Thus, when using C it can be easier to determine the window size, while the performance is better than A when the user's bid price is similar to or lower than the mean price.

Figure 12 shows a comparison of A and C checkpointing in terms of the task completion time. Again, C provides better performance when the window size is longer. Heuristic A provides similar performance but is inconsistent for different window sizes.

### 3.2.3 *Impact of work migration schemas*

Our investigation here is aimed at understanding the impact of work migration on high-CPU instance types. As described in Section 2.4, changing the instance type after a failure can reduce the waiting time to re-acquire

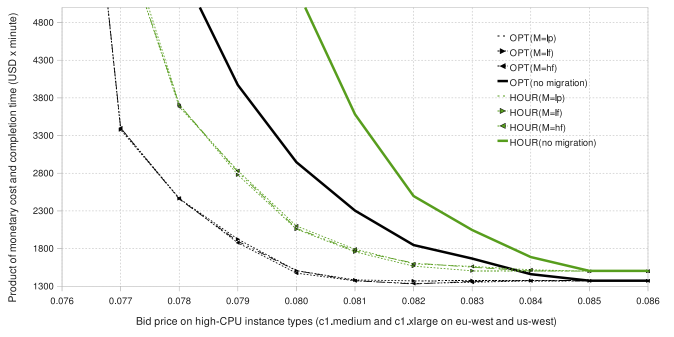

Fig. 13. Product of monetary cost and completion time of the optimal and hour checkpointing according to migration policies

resources. In Fig. 3, we observed that combination of availability on several instances can increase the fraction of total availability even when a user makes low bid prices among them. In this evaluation, we use the high-CPU instance types (c1.medium and c1.xlarge) of eu-west and us-west zones.

Table 4 shows a description of the investigated high-CPU instance types. We did not take into account the US-east instances, because their ranges of bid price are significantly different from EU-west and US-west. The extra-large (xlarge) instance types have 4 times more computation power, mean price, and the range of bid prices. In this simulation, we used the range of bid prices of eu-west.medium instances, and used normalized bid price for the other instance types with a factor of number of CPUs. For instance, us-west.xlarge type has 20 CPUs, so the normalized range of bid price is starting from $0.304/4 = 0.076$ USD.

TABLE 4
Prices of high-CPU linux instance types (Jan. 11th to Jul. 10th 2010)

| Instance type (high-CPU, linux) | # of CPUs (EC2 Units) | Lowest bid price | Highest bid price | Mean price |
|---|---|---|---|---|
| eu-west.medium | 5 | 0.076 | 0.103 | 0.080 |
| eu-west.xlarge | 20 | 0.304 | 0.450 | 0.322 |
| us-west.medium | 5 | 0.076 | 0.084 | 0.080 |
| us-west.xlarge | 20 | 0.304 | 0.750 | 0.321 |
| us-east.medium | 5 | 0.057 | 0.170 | 0.061 |
| us-east.xlarge | 20 | 0.226 | 0.690 | 0.242 |

Figure 13 represents the product of cost and time of the OPT and HOUR schemas with different migration policies. In this result, it is hard to see a significant difference among the migration policies, but we can conclude that migration is extremely helpful for reducing the product of cost and time in both OPT and HOUR checkpointing schemas.

Figure 14 compares the performance of the adaptive checkpointing schemas with the OPT and HOUR, in terms of the cost, time, and their product. From the results in Figs. 13 and 14, we see that migration can reduce the cost×time product, by reducing the task completion time, which is a verification of the idea of combining availability (see Fig. 3).

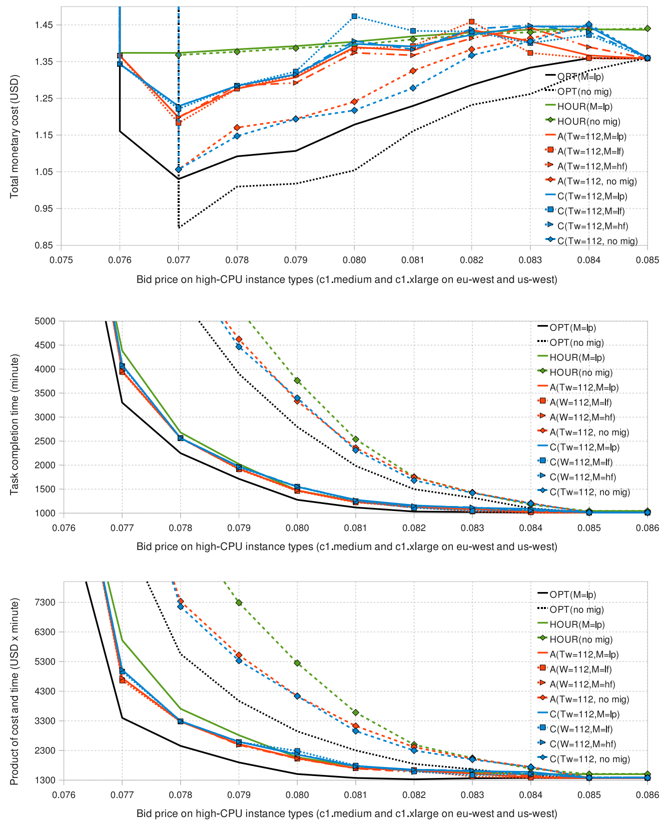

Fig. 14. Adaptive checkpointing with migration

TABLE 5
Cost, time, and their product on low price bidding (USD 0.078) normalized by normal on-demand instance type (eu.west.c1.medium)

| Checkpointing policy | Cost | Time | Cost×Time |
|---|---|---|---|
| (normalized by) | (USD 3.23) | (1,000 mins) | (3,230) |
| OPT($M_{LP}$) | 0.304 | 1.886 | 0.573 |
| OPT($no\ mig$) | 0.283 | 5.258 | 1.488 |
| HOUR($M_{LP}$) | 0.429 | 2.598 | 1.114 |
| HOUR($no\ mig$) | 0.427 | 7.983 | 3.409 |
| A($T_w$=112, $M_{LP}$) | 0.360 | 2.205 | 0.794 |
| A($T_w$=112, $no\ mig$) | 0.337 | 6.237 | 2.102 |
| C($T_w$=112, $M_{LP}$) | 0.372 | 2.260 | 0.841 |
| C($T_w$=112, $no\ mig$) | 0.328 | 6.146 | 2.016 |

Table 5 represents the normalized results compared with the normal on-demand instance type, where the on-demand instance is not a spot instance, but is assumed to be available 100% without interruption from Amazon. There, lower values are better. We compared the results in Fig. 14 with the case of using the on-demand "eu-west.c1.medium" instance type. The results show that migration significantly reduces the time, but it consumes a little bit more monetary cost. Thus, using migration can provide better product values, and as we observe in the results, the adaptive checkpointing policies have less than $1$ as the product of cost and time.

### 3.2.4 Low price bidding on 42 instance types

In the remaining subsections, we do a comprehensive comparison of the checkpointing policies across all 42 instance types, fixing the bid price parameter. In this subsection, we focus on bids with a relatively low price. In the following subsection we focus on bids at the mean price.

Table 6 shows the normalized product of the total price and the task completion time when a user bids a relatively low price based on the past price history. Here, the low bid price is calculated as $(mean\_price - lowest\_price)/2$, where the $lowest\_price$ is the lowest value on the range of bid prices. In this result, we observe that checkpointing policies affect the real price significantly. In particular, using hourly checkpointing (HOUR) has $30 \sim 150\%$ overhead than the optimal, while the best checkpointing policies on each instance type has $10 \sim 35\%$ overhead. Among the checkpointing policies, the C(112) shows the best performance on 9 Linux instance types, while the other C policies also show good results.

### 3.2.5 Mean price bidding on 42 instance types

Table 7 shows the normalized cost$\times$time product when a user bids the mean price based on the past price history. Similar to the results in Table 6, the checkpointing policies have significant impact on the total overhead on executing tasks on volatile cloud resources. The C(112) still shows better performance than others on the mean price bidding, except for the case "us-east.m1.large" instance type. This instance has quite different behavior before July after, and there is less price fluctuations, and thus, just using an hourly checkpointing can provide reasonable performance. In our previous work [8], we did not take into account the current price, nor the memoryfull property of failure arrivals. Thus, in the previous work, there was $30 \sim 45\%$ overhead on the best strategy for each instance type, but in the current work, we have only $8 \sim 35\%$ overhead compared with the optimal case.

### 3.2.6 Impact of different task lengths

Table 8 presents the impact of the different task lengths on the performance. In case of computing shorter tasks, they have low overhead because there are less chances to get the failures compared with longer tasks. As we observe in the results, task length does not have significant impact on the performance of the checkpointing policies. Therefore, we can use the task length used in this work ($1,000$ minutes) can be used as a reference value to determine the best checkpointing policy with given spot instance type, and bid price.

### 3.2.7 Policy comparison and result summary

Table 9 shows the best checkpointing policies for all $42$ types of spot instances. We observe that the adaptive checkpointing aware of the current price performs best for most cases. C(112) is the best policy on 37 different cases (including both low and mean price bidding). However, in some instances, the adaptive checkpointing policies were worse than the HOUR and EDGE policies (4 cases).

Summarizing, we observe that checkpointing can significantly affect both the task completion time and the total monetary cost. We found that using hourly checkpointing can reduce costs significantly in the presence of failures. But, we also found that simple policies (HOUR and EDGE) work well in very few instance types. By comparing our previous results [8] and the current work, we have found that taking into account the current price and optimizing the amount of past traces for modeling failure probabilities (in case of adaptive checkpointing) can significantly reduce the performance gap between the optimal and the tested checkpointing policies.

## 4 Related Work

Related work on Cloud Computing include multiple aspects: economics, management services, and fault-tolerant middleware. Several previous works focus on the economics of Cloud Computing [11]–[15]. However, these works assume a static pricing model for EC2's dedicated on-demand instances. They evaluate the cost-benefit of Cloud Computing compared to self-built, dedicated infrastructures such as traditional Grids or ISP's. The authors focus on different types of applications including task parallel, message passing, and data-intensive applications.

Several services for monitoring and managing cloud applications exist [16]–[18], but these services currently do not consider cloud costs that vary dynamically over time. For instance, RightScale [18] is a third party cloud computing broker that provides management services for clouds, such as EC2. They provide several software tools that reduce the complexity of managing and monitoring cloud computing resources. However, they still do not have any service for efficiently utilizing the spot instances on the Amazon EC2. Instead, the users of spot instances have to manage spot instance costs and reliability manually and individually.

Several middleware currently deployed over Clouds have fault-tolerance mechanisms [19]–[21], but these mechanisms currently are not cost-aware. For instance, Map-Reduce [19] and Condor [20] are intrinsically fault-tolerant, but how to conduct fault-tolerance in a cost-effective way has not been addressed. In particular, checkpointing has been well-studied, but previous studies have not taken into account variable resource costs. In [22], A. Duda studied the optimal placement of a checkpoint if the performance overhead is constant. In [10], Yi et al. proposed an adaptive checkpointing scheme

TABLE 6
Normalized price×time product for execution on low price bidding (digits in A() and C() are the window sizes $T_w$)

| Linux instance | Bid price | NONE | HOUR | EDGE | A(28) | A(56) | A(84) | A(112) | C(28) | C(56) | C(84) | C(112) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| eu-west.c1.medium | 0.077 | - | 2.590 | - | 1.392 | 1.352 | 1.358 | 1.347 | 1.252 | **1.240** | 1.241 | 1.241 |
| eu-west.c1.xlarge | 0.312 | - | 1.966 | - | 1.363 | 1.358 | 1.368 | 1.388 | 1.302 | 1.287 | **1.270** | 1.281 |
| eu-west.m1.large | 0.155 | - | 2.041 | - | 1.419 | 1.426 | 1.426 | 1.425 | 1.291 | **1.250** | 1.254 | 1.254 |
| eu-west.m1.small | 0.039 | - | 1.857 | - | 1.338 | 1.381 | 1.381 | 1.427 | **1.253** | 1.260 | 1.258 | 1.259 |
| eu-west.m1.xlarge | 0.313 | - | 1.946 | 363.1 | 1.509 | 1.408 | 1.417 | 1.393 | 1.573 | 1.413 | 1.344 | **1.333** |
| eu-west.m2.2xlarge | 0.545 | - | 1.768 | - | 1.407 | 1.413 | 1.390 | 1.446 | 1.364 | 1.340 | 1.328 | **1.323** |
| eu-west.m2.4xlarge | 1.092 | - | 1.891 | - | **1.378** | 1.468 | 1.494 | 1.510 | 2.377 | 1.807 | 1.667 | 1.613 |
| us-east.c1.medium | 0.058 | 96.77 | 2.012 | - | 1.353 | **1.266** | 1.285 | 1.469 | 1.286 | 1.319 | 1.344 | 1.347 |
| us-east.c1.xlarge | 0.241 | - | 1.411 | 13.55 | 1.379 | 1.429 | 1.408 | 1.439 | 1.497 | 1.293 | 1.270 | **1.267** |
| us-east.m1.large | 0.141 | - | 1.121 | 2.421 | 1.326 | 1.105 | 1.101 | **1.100** | 2.120 | 2.112 | 2.112 | 2.119 |
| us-east.m1.small | 0.030 | - | 1.457 | - | 1.293 | 1.352 | 1.389 | 1.330 | 1.288 | 1.284 | 1.260 | **1.256** |
| us-east.m1.xlarge | 0.236 | - | 1.795 | 304.3 | 1.540 | 1.501 | 1.536 | 1.478 | 1.722 | 1.527 | 1.450 | **1.446** |
| us-east.m2.2xlarge | 0.411 | - | 1.675 | - | 1.495 | 1.481 | 1.446 | **1.388** | 1.903 | 1.554 | 1.486 | 1.487 |
| us-east.m2.4xlarge | 0.825 | - | 1.581 | 79.87 | 1.372 | 1.348 | 1.391 | **1.334** | 3.327 | 2.278 | 1.728 | 1.566 |
| us-west.c1.medium | 0.077 | - | 1.631 | - | 1.508 | 1.381 | 1.602 | 1.695 | 1.401 | **1.288** | 1.290 | 1.312 |
| us-west.c1.xlarge | 0.311 | - | 1.373 | - | 1.489 | 1.446 | 1.502 | 1.459 | 1.673 | 1.437 | 1.420 | **1.364** |
| us-west.m1.large | 0.156 | - | 1.582 | 262.1 | 1.459 | 1.421 | 1.468 | 1.425 | 1.377 | 1.293 | 1.292 | **1.279** |
| us-west.m1.small | 0.039 | - | 1.353 | - | 1.382 | 1.320 | 1.313 | 1.393 | 1.280 | 1.258 | **1.245** | 1.262 |
| us-west.m1.xlarge | 0.311 | - | 1.603 | 52.54 | 1.462 | 1.474 | 1.441 | 1.441 | 1.820 | 1.651 | 1.466 | **1.380** |
| us-west.m2.2xlarge | 0.545 | - | 1.576 | 90.30 | 1.489 | 1.474 | 1.486 | 1.442 | 2.202 | 1.494 | 1.400 | **1.364** |
| us-west.m2.4xlarge | 1.091 | - | 1.654 | - | **1.382** | 1.469 | 1.464 | 1.479 | 3.021 | 1.974 | 1.562 | 1.554 |

TABLE 7
Normalized price×time product for execution on mean price bidding (digits in A() and C() are the window sizes $T_w$)

| Linux instance | Bid price | NONE | HOUR | EDGE | A(28) | A(56) | A(84) | A(112) | C(28) | C(56) | C(84) | C(112) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| eu-west.c1.medium | 0.080 | - | 1.959 | 208.6 | 1.478 | 1.521 | 1.533 | 1.508 | **1.296** | 1.297 | 1.330 | 1.316 |
| eu-west.c1.xlarge | 0.322 | - | 1.607 | 45.14 | 1.343 | 1.372 | 1.476 | 1.481 | 1.443 | 1.384 | 1.243 | **1.239** |
| eu-west.m1.large | 0.160 | - | 1.915 | 55.34 | 1.483 | 1.461 | 1.449 | 1.462 | 1.368 | **1.304** | 1.328 | 1.314 |
| eu-west.m1.small | 0.040 | - | 1.746 | 435.7 | 1.442 | 1.456 | 1.434 | 1.439 | **1.331** | 1.346 | 1.364 | 1.362 |
| eu-west.m1.xlarge | 0.320 | - | 1.732 | 92.15 | 1.385 | 1.386 | 1.409 | 1.394 | 1.507 | 1.391 | 1.314 | **1.273** |
| eu-west.m2.2xlarge | 0.560 | - | 1.645 | 71.57 | 1.399 | 1.374 | 1.371 | 1.396 | 1.782 | 1.640 | 1.466 | **1.375** |
| eu-west.m2.4xlarge | 1.121 | - | 1.777 | 27.73 | **1.489** | 1.530 | 1.553 | 1.535 | 2.750 | 2.036 | 1.859 | 1.742 |
| us-east.c1.medium | 0.060 | - | 1.490 | 94.12 | 1.395 | 1.429 | 1.425 | 1.420 | 1.425 | 1.361 | **1.332** | 1.334 |
| us-east.c1.xlarge | 0.287 | 1.627 | 1.113 | 1.203 | 1.243 | 1.169 | 1.169 | **1.089** | 1.398 | 1.308 | 1.308 | 1.277 |
| us-east.m1.large | 0.142 | 2.442 | 1.121 | 2.421 | 1.326 | 1.105 | 1.101 | **1.100** | 2.124 | 2.111 | 2.111 | 2.117 |
| us-east.m1.small | 0.032 | 1.343 | 1.119 | 1.099 | 1.223 | 1.100 | 1.101 | 1.102 | **1.085** | 1.096 | 1.095 | 1.093 |
| us-east.m1.xlarge | 0.245 | - | 1.376 | 11.28 | **1.355** | 1.424 | 1.438 | 1.401 | 1.863 | 1.524 | 1.407 | 1.405 |
| us-east.m2.2xlarge | 0.420 | - | 1.443 | 34.39 | 1.411 | 1.402 | 1.419 | 1.412 | 1.739 | 1.547 | 1.411 | **1.400** |
| us-east.m2.4xlarge | 0.864 | 18.85 | **1.231** | 3.259 | 1.256 | 1.310 | 1.285 | 1.282 | 3.010 | 1.937 | 1.798 | 1.581 |
| us-west.c1.medium | 0.080 | 316.2 | 1.313 | 20.21 | 1.359 | 1.372 | 1.402 | 1.392 | 1.330 | 1.270 | 1.269 | **1.242** |
| us-west.c1.xlarge | 0.320 | 59.46 | **1.297** | 35.03 | 1.432 | 1.377 | 1.385 | 1.369 | 1.768 | 1.465 | 1.398 | 1.385 |
| us-west.m1.large | 0.160 | 90.55 | 1.320 | 37.15 | 1.373 | 1.381 | 1.384 | 1.418 | 1.438 | 1.302 | 1.304 | **1.301** |
| us-west.m1.small | 0.040 | 380.0 | 1.365 | 34.98 | 1.334 | 1.373 | 1.306 | 1.316 | 1.345 | 1.262 | 1.251 | **1.251** |
| us-west.m1.xlarge | 0.320 | 844.3 | 1.379 | 20.17 | 1.451 | 1.411 | 1.402 | 1.462 | 1.731 | 1.529 | 1.420 | **1.340** |
| us-west.m2.2xlarge | 0.560 | 129.3 | 1.334 | 13.05 | 1.487 | 1.387 | 1.328 | **1.304** | 2.657 | 1.693 | 1.567 | 1.511 |
| us-west.m2.4xlarge | 1.120 | 51.61 | 1.373 | 11.97 | **1.362** | 1.379 | 1.401 | 1.366 | 3.186 | 1.935 | 1.499 | 1.403 |

which provides adaptive taking point decision function when the cost of checkpointing changes over time. Their results apply under the assumption that failures occur according to the *Poisson* process. In contrast, we use the probability density function which is calculated from the previous traces of spot instances.

There are several challenges related to checkpointing in context of unreliable resources such as spot instances. The first one is finding the relationship between past and future failures or availability for proactive checkpointing. Much work exists on finding correlations and dependence between failure events [23]–[26]. Another challenge is using an efficient checkpointing method for minimizing the expected execution time in the presence of failures. This also has been the subject of previous work described in [10], [22], [27], [28]. A new aspect is understanding the impact of checkpointing methods on the spot instances for reducing both the monetary costs and the task's total execution time.

In November 2010, Enomaly Inc. announced SpotCloud [3], in which someone can be both as a buyer and a seller. Using the SpotCloud, buyers can get resources by bidding on the spot markets, and sellers can get profit by registering their available computing resources. Thus, in the middle, SpotCloud manages the spot price fluctuations based on the supply and demand relationship.

TABLE 8
Normalized price×time product for execution on eu-west-1.linux.m1.large instance type (digits in A() and C() are the window sizes $T_w$)

| Task length | Bid price | NONE | HOUR | EDGE | A(28) | A(56) | A(84) | A(112) | C(28) | C(56) | C(84) | C(112) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 200 minutes | 0.155 | - | 1.198 | 331.4 | 1.226 | 1.224 | 1.225 | 1.224 | 1.198 | **1.186** | 1.203 | 1.203 |
| 500 minutes | 0.155 | - | 1.885 | - | 1.350 | 1.353 | 1.351 | 1.348 | 1.299 | **1.225** | 1.228 | 1.228 |
| 1,000 minutes | 0.155 | - | 2.041 | - | 1.419 | 1.426 | 1.426 | 1.425 | 1.291 | **1.250** | 1.254 | 1.254 |
| 2,000 minutes | 0.155 | - | 2.169 | - | 1.469 | 1.470 | 1.485 | 1.481 | 1.297 | **1.271** | 1.284 | 1.283 |
| 5,000 minutes | 0.155 | - | 2.099 | - | 1.423 | 1.432 | 1.451 | 1.447 | 1.283 | **1.273** | 1.279 | 1.280 |
| 200 minutes | 0.160 | 54.58 | 1.640 | 17.82 | 1.280 | 1.267 | 1.264 | 1.255 | 1.234 | **1.216** | 1.222 | 1.217 |
| 500 minutes | 0.160 | - | 1.789 | 47.90 | 1.445 | 1.440 | 1.425 | 1.421 | 1.373 | **1.300** | 1.327 | 1.309 |
| 1,000 minutes | 0.160 | - | 1.915 | - | 1.483 | 1.461 | 1.449 | 1.462 | 1.368 | **1.304** | 1.328 | 1.314 |
| 2,000 minutes | 0.160 | - | 1.920 | - | 1.549 | 1.525 | 1.516 | 1.490 | 1.405 | **1.306** | 1.339 | 1.321 |
| 5,000 minutes | 0.160 | - | 1.950 | - | 1.630 | 1.545 | 1.546 | 1.527 | 1.366 | **1.328** | 1.354 | 1.345 |

TABLE 9
Best checkpointing policy for each spot instance type on low and mean price bidding, in terms of price×time product

| Instance zone | eu-west.linux | | eu-west.windows | | us-east.linux | | us-east.windows | | us-west.linux | | us-west.windows | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bid price | low | mean | low | mean | low | mean | low | mean | low | mean | low | mean |
| c1.medium type | C(56) | C(28) | C(112) | C(112) | A(56) | C(84) | C(84) | EDGE | C(56) | C(112) | C(84) | C(112) |
| c1.xlarge type | C(84) | C(112) | C(112) | C(112) | C(112) | A(112) | C(112) | A(112) | C(112) | HOUR | C(112) | A(84) |
| m1.large type | C(56) | C(56) | C(112) | C(112) | A(112) | A(112) | C(112) | C(112) | C(112) | C(112) | C(112) | C(112) |
| m1.small type | C(28) | C(28) | C(84) | C(28) | C(112) | C(28) | A(112) | A(84) | C(84) | C(112) | C(112) | C(84) |
| m1.xlarge type | C(112) | C(112) | C(112) | C(112) | C(112) | A(28) | C(112) | A(28) | C(112) | C(112) | A(28) | A(56) |
| m2.2xlarge type | C(112) | C(112) | C(84) | C(112) | A(112) | C(112) | C(112) | C(112) | C(112) | A(112) | HOUR | A(56) |
| m2.4xlarge type | A(28) | A(28) | A(28) | A(28) | A(112) | HOUR | A(56) | A(56) | A(84) | A(28) | A(56) | A(28) |

## 5 Conclusions and Future Work

We investigated several different approaches to reduce both monetary cost and task completion time of computations using Amazon EC2's spot instances for resource provisioning. Our main contributions are as follows:

1) We determine analytically the expected failure recovery time when a checkpointing is taken versus skipped when the distribution of availability is any *general* distribution. We prove the correctness of our formulas for the expected recovery time if a checkpoint is taken or skipped. The latter can be used to determine adaptively whether a periodic checkpoint should be skipped.
2) We evaluate several heuristics for checkpointing that use various predictive mechanisms to deal with dynamically varying prices. In our evaluation, we use simulation based on real price traces of Amazon's spot instances. We find that adaptive checkpointing that takes into account the current price most often performs best and reduces monetary costs the most.
3) We evaluate the time window used by these different checkpointing heuristics. We find that with adaptive checkpointing based on the current price, a longer window size (e.g., 112 days) often minimizes total monetary costs. Performance is best with this checkpointing schema when the user's bid price is less than or equal to the mean price.
4) We evaluate several heuristics for work migration across different types of spot instances. We find that work migration can reduce execution time by more than a factor of 2.5 compared to cases where it is not used. The cost is only slightly higher compared when not using migration. The heuristic chosen does not make a significant difference in the savings of execution time.

Our future work will include determining the appropriate points for task recovery to exploit non-paying partial hours on spot instances. We are also interested in identifying correlation between past and current prices, between instance types, and between price fluctuations. Another aspect of price modeling is identification and exploiting of calendar-based effects such as price differences between week days and weekends or changes of day / night prices in relation to the time zone. We will investigate how to develop robust prediction methods to minimize monetary costs and completion times under these schemas. Another thread of work is a comparative evaluation of our strategies in a different scenario scenario than Amazon EC2. It is also useful work to find the appropriate management for the spot prices from Cloud vendor's perspective.

## Acknowledgments

This work is carried out in part under the EC project eXtreemOS (FP6-033576) and the ANR project Clouds@home (ANR-09-JCJC-0056-01). The authors would like to thank the anonymous reviewers for suggestions on improving the readability of this work.

## Reproducibility of Results

All data used in this study, the full source code of the simulator and additional results are available at the following URL:

http://spotckpt.sourceforge.net

## REFERENCES

[1] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a Market Economy to Provision Compute Resources Across Planet-wide Clusters," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'09)*, 2009.
[2] Amazon EC2 Spot Instances, *http://aws.amazon.com/ec2/spot-instances/*, 2010.
[3] "SpotCloud - Cloud Capacity Clearing House / Spot Market," http://spotcloud.com.
[4] Y. Yang and H. Casanova, "Umr: A multi-round algorithm for scheduling divisible workloads." in *IPDPS*, 2003, p. 24.
[5] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *MASCOTS'10*, August 2010.
[6] "The berkeley open infrastructure for network computing," http://boinc.berkeley.edu/.
[7] Amazon Simple Storage Service FAQs, *http://aws.amazon.com/s3/faqs/*, 2010.
[8] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud," in *The 3rd International Conference on Cloud Computing (CLOUD'10)*, July 2010, pp. 236–243.
[9] S. Yi and D. Kondo, "How checkpointing can reduce cost of using clouds?" in *The 3rd EU-Korea Conference on Science and Technology (EKC'10)*, August 2010.
[10] S. Yi, J. Heo, Y. Cho, and J. Hong, "Taking point decision mechanism for page-level incremental checkpointing based on cost analysis of process execution time," *Journal of Information Science and Engineering*, vol. 23, no. 5, pp. 1325–1337, September 2007.
[11] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *18th International Heterogeneity in Computing Workshop*, Rome, Italy, May 2009. [Online]. Available: http://mescal.imag.fr/membres/derrick.kondo/pubs/kondo_hcw09.pdf
[12] A. Andrzejak, D. Kondo, and D. P. Anderson, "Exploiting non-dedicated resources for cloud computing," in *12th IEEE/IFIP Network Operations & Management Symposium (NOMS 2010)*, Osaka, Japan, Apr 19–23 2010.
[13] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for Science Grids: a Viable Solution?" in *Data-Aware Distributed Computing Workshop (DADC)*, 2008.
[14] S. Garfinkel, "Commodity grid computing with amazons s3 and ec2," in *login*, 2007.
[15] E. Deelman, S. Gurmeet, M. Livny, J. Good, and B. Berriman, "The Cost of Doing Science in the Cloud: The Montage Example," in *Proc. of Supercomputing'08, Austin*, 2008.
[16] CloudStatus, *http://www.cloudstatus.com/*, 2010.
[17] CloudKick: Simple, powerful tools to manage and monitor cloud servers, *https://www.cloudkick.com/*, 2010.
[18] RightScale: Cloud Computing Management Platform, *http://www.rightscale.com/*, 2010.
[19] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.
[20] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)*, 1988.
[21] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes," in *Proceedings of SC'02*, 2002.
[22] A. Duda, "The effects of checkpointing on program execution time," *Information Processing Letters*, vol. 16, no. 1, pp. 221–229, Jul. 1983.
[23] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *SC'07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–12.
[24] B. Javadi, D. Kondo, J. Vincent, and D. Anderson, "Mining for availability models in large-scale distributed systems: A case study of seti@home," in *17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2009.
[25] D. Kondo, A. Andrzejak, and D. P. Anderson, "On correlated availability in internet distributed systems," in *IEEE/ACM International Conference on Grid Computing (Grid)*, Tsukuba, Japan, 2008.
[26] A. Andrzejak, P. Domingues, and L. M. Silva, "Predicting machine availabilities in desktop pools," in *10th IEEE/IFIP Network Operations & Management Symposium (NOMS 2006)*, Vancouver, Canada, April 3–7 2006, pp. 1–4.
[27] J. S. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 10, pp. 972–986, October 1998.
[28] S. Yi, J. Heo, Y. Cho, and J. Hong, "Adaptive page-level incremental checkpointing based on expected recovery time," in *2006 ACM Symposium on Applied Computing (ACM SAC'06)*, April 2006, pp. 1472–1476.

**Sangho Yi** received the B.E. degree in Electrical Engineering from Korea University, Seoul, Korea in 2003 and the Ph.D. degree in School of Computer Science and Engineering, Seoul National University, Seoul, Korea in 2008. He was a post-doctoral researcher in Seoul National University, Korea, and INRIA, France, since 2011. He has been with DMC Research and Development Center, Samsung Electronics, Korea, where currently he is a senior research engineer. His research interests include embedded operating systems, fault tolerance, and volunteer computing.
Homepage: http://antiroot.net

**Artur Andrzejak** is a W3-professor at Ruprecht-Karls-University of Heidelberg and leads there the Parallel and Distributed Systems research group. He received a PhD degree in computer science from ETH Zurich (2000) and a habilitation degree from Freie Universitaet Berlin (2009). He was a postdoctoral researcher at the HP Labs Palo Alto 2001 to 2002, a researcher at Zuse-Institute-Berlin 2003 to 2009 and a deputy department head at I2R Singapore in 2010. His research interests include modeling and dependability of distributed systems, management of data centers, utility/cloud computing, and applications of data mining.
Hompage: http://pvs.ifi.uni-heidelberg.de/team/aa

**Derrick Kondo** is a tenured research scientist at INRIA, France. He received his Bachelor's at Stanford University in 1999, and his Master's and Ph.D. at the University of California at San Diego in 2005, all in computer science. His general research interests are in the areas of reliability, fault-tolerance, statistical analysis, scheduling and resource management for parallel and distributed systems. His research projects are supported by national, European, and industrial grants. He is co-founder of the Failure Trace Archive, which serves as a public repository of failure traces and algorithms for distributed systems.
Homepage: http://mescal.imag.fr/membres/derrick.kondo