Cost-efficient and Resilient Job Life-cycle Management on Hybrid Clouds

Hsuan-Yi Chu
Computer Science Department
University of Southern California
Los Angeles, CA 90089, U.S.A.
Email: hsuanyi@usc.edu

Yogesh Simmhan
Supercomputer Education and Research Center
Indian Institute of Science
Bangalore 560012, India
Email: simmhan@serc.iisc.in

***Abstract*—Cloud infrastructure offers democratized access to on-demand computing resources for scaling applications beyond captive local servers. While on-demand, fixed-price Virtual Machines (VMs) are popular, the availability of cheaper, but less reliable, *spot VMs* from cloud providers presents an opportunity to reduce the cost of hosting cloud applications. Our work addresses the issue of effective and economic use of hybrid cloud resources for planning job executions with deadline constraints. We propose strategies to manage a job's life-cycle on spot and on-demand VMs to minimize the total dollar cost while assuring completion. With the foundation of *stochastic optimization*, our reusable table-based algorithm (RTBA) decides when to instantiate VMs, at what bid prices, when to use local machines, and when to checkpoint and migrate the job between these resources, with the goal of *completing the job on time and with the minimum cost*. In addition, three simpler heuristics are proposed as comparison. Our evaluation using historical spot prices for the Amazon EC2 market shows that RTBA on an average reduces the cost by 72%, compared to running only on on-demand VMs. It is also robust to fluctuations in spot prices. The heuristic, H3, often approaches RTBA in performance and may prove adequate for *ad hoc* jobs due to its simplicity.**

I. Introduction

Cloud computing has become a first-class resource platform for application service providers to host their services and complement their local computing infrastructure. The lower barrier to entry, both in the initial infrastructure cost and the service-based middleware tooling, has made clouds popular for enterprise and scientific applications [4]. Public cloud service providers such as Amazon Web Services (AWS) and Microsoft's Azure offer access to compute resources through Virtual Machine (VM) instances with diverse capabilities, and to persistent storage services.

Cloud providers follow a *pay-as-you-go* pricing model, where customers pay for the resources they acquire on-demand, in well-defined time and resource increments (e.g. hourly increments, small/medium/large VMs), using a *static price* list. Hence acquiring and releasing resources elastically is important to maximize the utility. While this elasticity is easy to leverage for repetitive, stateless jobs such as web services, larger applications can use this elasticity to pick a VM of the right size and use it exclusively rather than compete with other jobs on the same VM. This puts the focus

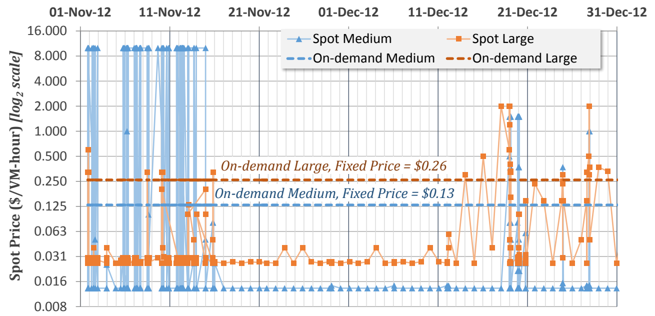
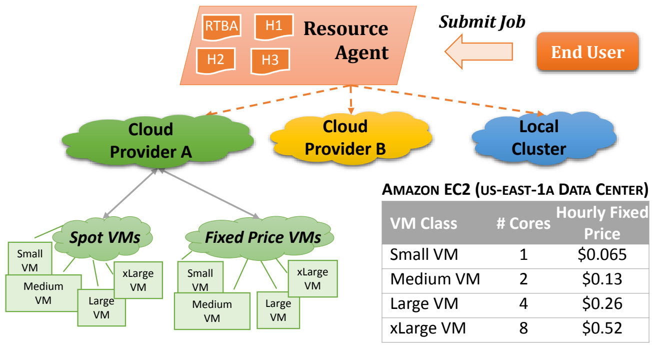


Figure 1: Amazon EC2 spot prices of medium and large instances at *us-east-1a* data center in Nov and Dec, 2012. *Fixed-price* on-demand VMs are also shown for comparison. Note that the spot prices of the medium VMs have significant fluctuations in Nov while being more stable in Dec.

on resource allocation for a single job rather than scheduling multiple jobs on few resources.

Recently, cloud providers such as Amazon are offering *spot priced* VMs. These are typically spare data center capacity offered at low prices to enhance the data center's resource utilization. Customers place their *bids* for VMs on the spot market at a price level of their choosing [1]. The cloud provider periodically publishes the actual spot prices at which they will offer these VMs. Customers whose bids are greater than the published spot prices will be given the VMs. The cloud provider can vary the spot prices over time, sometimes within minutes. Customers who have acquired a spot VM pay the current spot price as long as it is less than their bid price. However, when the spot price increases beyond their bid, the VM is immediately revoked from the customer. This is termed as an "out-of-bid event", and *all the applications, state and data present in that VM are lost*. Thus, while spot VMs are usually cheaper than fixed-price VMs and perform equally well, they are susceptible to revocation due to the pricing mechanism of the cloud provider which in itself is not a true market-based model. For instance, in Fig. 1, the prices for medium-sized spot VMs are usually $0.014/hour as compared to $0.13/hour for the fixed-price ones, but spike to over $8/hour, thus revoking literally all medium spot VMs. As a result, long running applications that pay a cumulative higher cost for fixed-price VMs and



AMAZON EC2 (US-EAST-1A DATA CENTER)

VM Class	# Cores	Hourly Fixed Price
Small VM	1	$0.065
Medium VM	2	$0.13
Large VM	4	$0.26
xLarge VM	8	$0.52

Figure 2: The resource agent accepts jobs from users with specified deadline constraints. The agent plans strategies to provision resources and manage checkpoint/migration on a hybrid cloud to finish the job by the deadline with the minimum rental cost.

would benefit from using cheaper spot VMs will need to be resilient, using simple *ab initio* restart after losing their prior work, or through checkpoint and resume mechanisms.

Problem. We refer to the diverse and distributed computing resources available to a customer: fixed-price on-demand VMs [1], spot VMs, and limited capacity on "free" local servers or private clouds they own, as a *hybrid cloud* environment (Fig. 2). Cloud VMs further have different VM sizes and pricing based on computing capabilities. A job with an expected runtime (core-hours) and a deadline for completion has to make the best use of the hybrid cloud to minimize total cost while ensuring completion within the deadline. Our proposed resource agent *makes decisions on the life-cycle of this job* – which resource to start on, at what price to bid for spot VMs, when to checkpoint, when to migrate and resume, and when to restart – to meet the objectives. In particular, the challenge arises from intelligently using spot VMs at the right bid price to balance total cost against the deadline, and mitigating the impact of out-of-bid events through job life-cycle management. This is a novel problem that confronts real applications, with the ability to maximize the economic utility of hybrid resources.

Results. With the foundation of *stochastic optimization*, we propose a **Reusable Table-Based Algorithm (RTBA)** which facilitates the resource agent's decision-making for a given job's life-cycle on hybrid resources. The algorithm is reduced to *Bellman's equation* to meet the deadline and cost minimization objective, proving the *minimum* expected resource cost. We also propose three simpler heuristics (**H1–H3**) for making these decisions that may prove adequate and cost effective for *ad hoc* jobs. Our simulation study using different job characteristics and 5 months of Amazon EC2 Spot VM pricing demonstrates the following:

- Our RTBA approach reduces the resource cost by an average of 72%, compared to running purely on fixed-price on-demand VMs, for the job workload we study.
- All three heuristics H1–H3 offer less cost savings than RTBA with average improvements of 43%, 53% and 67%, respectively, compared to purely fixed-price VMs. The heuristic H3 often performs close to RTBA and may be more practical due to its simplicity.
- RTBA uses historical spot prices to develop a price model which offers spot price estimates [2]. We empirically show the robustness of RTBA to different weakly reliable price models. The RTBA's cost performance varies narrowly for different models but is always better than on-demand VMs and the heuristics.

The rest of the paper is structured as follows: we present related work in § II, define the hybrid cloud system and job model in § III, formalize the problem mathematically in § IV, develop optimal and heuristic-based solutions in § V and § VI, summarize the results of our simulation study in § VII, and offer our future directions and conclusions in § VIII.

II. RELATED WORK

Public cloud resources cost real money. Hence, from a cloud user's perspective, it is important to minimize *monetary cost*, which is distinct from the minimization of *resource usage* [24]. The non-uniform pricing of spot VMs offers the possibility of using more (but transiently cheaper) resources for a cost effective solution. As a result, existing resource minimization algorithms that may suffice for fixed-price VMs are inadequate for variable-priced spot VMs. Our work lies in the intersection of several research areas.

Hybrid Clouds. Cloud bursting is a term used when an organization uses in-house servers to support normal application needs and requests additional cloud resources during workloads spikes [6], [9], [17]. We consider an analogous but inverse model where jobs are primarily queued and run on clouds where they have the option of choosing from different VM types, but can also access limited off-cloud local servers. This predicates the use of clouds for *normal operations at scale* and sparse (but free) local resources as a backup. One distinction here is that, instead of an overhead to deploy jobs from local servers to the cloud, there is a cost for moving the job and its data out of the cloud.

Hosting scientific applications on Amazon EC2 has attracted significant attention [10], [13]. Some of them consider cost and deadline factors in provisioning resources [16], [15], [11]. For instance, [11] considers a similar problem of minimizing user-payments with guaranteed task deadlines. However, they do not consider spot VMs which can potentially be even cheaper but also requires novel approaches to ensure job resilience upon VM revocation.

Spot Pricing. The VM spot market was first introduced by Amazon in Dec, 2009 to exploit their unused data center capacity [1]. Spot VMs are usually much cheaper than their

[1]Both 'fixed-price on-demand' and 'spot' VMs are, in effect, acquired on-demand. Our terminology is based Amazon AWS's convention. We use the terms 'fixed-price on-demand' and 'on-demand' VMs interchangeably.

fixed-price equivalents though they offer the same performance, but without guaranteed resource availability. Spot prices are influenced more by Amazon's usage of on-demand VMs and spare capacity rather than the spot bids received or spot VM workloads that are submitted. Some studies have analyzed historical spot prices in an attempt to reverse-engineer the pricing logic. For instance, [2] speculates that the prices are determined artificially by a random reserve price algorithm. Moreover, [8] and [22] employ *Markov chain models* to characterize the spot prices. Importantly, while [23] shows the difficulty of predicting the spot prices, they also demonstrate the effectiveness of probabilistic spot price modeling and *stochastic optimization*, which serve as the basis of our RTBA approach.

Resilience. While bidding at a higher price can reduce the chance of revocation, the peak spot prices are way higher than fixed-prices of VMs (Fig. 1), making spot VMs inherently unreliable. Hence, bidding and fault-tolerance mechanisms are dual problems, and optimal decision-making should jointly consider these two dimensions. Checkpointing is a common approach to enhance resilience of jobs on unreliable resources. There is rich literature on checkpointing for HPC applications with the overhead being measured and modeled [7], [14]. Costs of checkpointing and migration may be asymmetric. Checkpointing cost may be trivial for some applications [22], [23] while dominating for others [19]. So checkpointing must be done judiciously to mitigate the overhead and additional resource costs.

[23] proposes an optimization problem which uses the *stochastic resource rental model* to plan resource provisioning on spot and fixed-price VMs. However, a key simplifying assumption in their model is that when an out-of-bid event occurs, the state of the job can be immediately migrated to an available on-demand VM. This is unrealistic as out-of-bid events are not notified in advance and occur instantaneously. Our proposed strategy addresses this lacuna. Specifically, once a VM is revoked, the job resumes from the last available checkpoint. In addition, the costs of checkpointing and migration are modeled [2], and the *startup time* for acquiring VMs is considered – these matter in practice. In other literature, out-of-bid events are handled either through checkpointing, migration and replication strategies [21], [20], or through bidding schemes [3], [18], [22]. Nevertheless, our work is unique since we co-design the bidding price and fault-tolerance decisions, which together minimize the cost while meeting the job's deadline requirement.

III. System and Job Models

A user submits a job J to our resource agent which makes life-cycle management decisions for it. Both the job and the agent are assumed to be in the cloud initially. The characteristics of the job J are described as a tuple $J(C, D, T_{dl})$, where D is the input data size, C is the number of core-hours of computation required, and T_{dl} is the deadline by which the job has to be completed. The followed notations are summarized in Table I. Further, we assume that the job speeds up linearly with the number of CPU cores present on a machine (i.e., it takes a 1-core small VM twice as long as a 2-core medium VM to finish a job).

[2]While both checkpointing and migration result in overhead times, migration can incur additional network charge if the job and its data are moved out of the cloud; often, in-cloud bandwidth is free while out-of-cloud bandwidth is billed.

Table I: Conventional Notations

Variable	**Description**
C	Job's Compute requirement (core-hours)
D	Job's input data size (bytes)
T_{dl}	Job's deadline constraint (hours)
$J(\cdot)^a$	Characteristics of the job J
t	Logical time index
a_t	Action taken at t
$\pi_{t'}$	Policy, a series of actions from $t = 0$ to $t = t'$
P_t^i	Spot price of the VM class i at t
$\mathbf{P}_t$	Set of spot prices for all VM classes at t
$c(t)$	Job's progress at t ^{b}
$c'(t)$	Job's residual compute after the last checkpoint
$x(t)$	Job's state at t
$A(x(t))$	Set of available actions for state $x(t)$
$v(t)$	Class of the VM that this job runs on at t
$\rho(v(t))$	CPU cores available in the VM of class $v(t)$
Δt	Index of the time-slot within a whole VM hour
$S_t(\cdot)$	Status of the job at t

[a]The notation $(\cdot)$ denotes the omitted argument of a tuple.
[b]This is measured as the residual computing that is still needed at t.

Based on offerings from contemporary cloud providers, the resource types we consider are: on-demand VMs, spot VMs and off-cloud local servers. VMs further offer multiple classes, differentiated by the number of cores (see Table in Fig. 2). Each class of on-demand VMs has a fixed hourly price, depending on the number of cores; fractional hours used are rounded up to whole hours. Spot VMs are priced on a different policy: (1) The price of each class varies independently, with time and orthogonal to the user's requests. Notably, the larger VMs with more cores are not necessarily costlier than the smaller one (see Fig. 1). (2) Spot VMs have to be bid for at a specified bid price. As long as the user's bid is higher than the current spot price, the VM is available to the user. (3) In case of out-of-bid events, where the spot price increases beyond the bid, the VM is immediately revoked and terminated with loss of all state; the partial hour used is not be billed. Otherwise, billing is by the hourly usage, as for fixed-price VMs. Both on-demand and spot VMs are associated with a start-up time – real clouds do not provision VMs instantaneously. Utilizing off-cloud local machines does not incur additional expense. However, transferring data from the cloud to the local machines costs network bandwidth that is billed proportional to the job's data size, D.

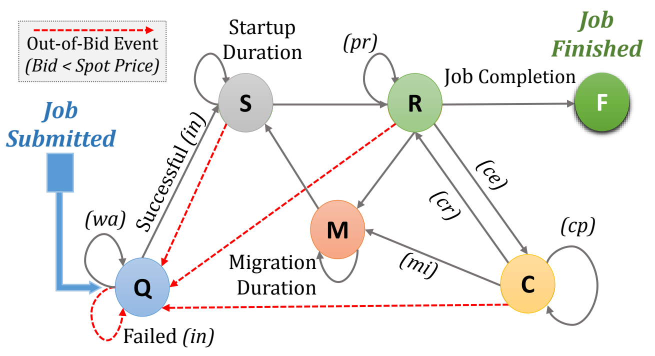


Figure 3: The state transition diagram for a job's life-cycle managed by the agent. Edges are labeled with actions taken by the agent from the set A, or forced by the cloud provider.

Both checkpointing and migration use a resource's time during which the job cannot progress. The times taken to perform checkpointing and migration are given as $T_C = \epsilon(c)$ and $T_M = \frac{D}{BW}$, respectively, where $\epsilon(c)$ is a function of the job-progress [3], D is the data size, and BW represents the *bandwidth* associated with the direction of data transfer (i.e., intra-cloud, local machine$\rightarrow$cloud, or cloud$\rightarrow$local machine), each of which can be different. While some prior works have ignored the cost of checkpointing [22], [23], in practice it can dominate application runtimes [19], particularly when applications are checkpointed more frequently if the *mean-time between failure* of the host machine is small. We incorporate a swathe of parameters in our framework to capture real system behavior and offer a wide solution space. These parameters can be pruned to balance accuracy, ability to collect the relevant information, and model complexity.

The time taken for checkpointing, migration and VM startup [4] can significantly undermine the economic benefits indirectly. Specifically, during these operations, a job cannot progress in its execution, and they cause the deadline to virtually "shrink". As can be imagined, an imminent deadline might preclude the use of cheaper spot VMs and force us to use more expensive resources, such as reliable on-demand VMs or larger sized VMs, to ensure on-time job completion. Thus, an intelligent agent, managing a job's life-cycle, must plan a series of actions that lead to the job's completion by deadline, and with the minimum total cost.

To begin with, we define the following terminology. The state of a job at time t is encoded by $x(t)$. A job can reside in one of the six states, detailed later. The *action*, a_t, applied at time t refers to the decision the agent makes at that instant, in response to the observed conditions. The available actions are given by the set $A = \{in, pr, ce, cp, cr, mi, wa\}$, which corresponds to initiating/bidding for an instance (*in*), processing the job (*pr*), starting to checkpoint (*ce*), continuing checkpointing (*cp*)[5], resuming from a checkpoint (*cr*), migrating the job (*mi*), and waiting for spot price to drop (*wa*). The *policy*, $\pi_{(T_{dl}-1)}$, is a sequence of actions $\{a_0, a_1, ..., a_{(T_{dl}-1)}\}$ from $t = 0, \ldots, (T_{dl} - 1)$ [5], which starts from the job's initial state, through a sequence of intermediate states, to the completion state.

We introduce the *transition diagram*, shown in Fig. 3, to describe the life-cycle of a job. It is assumed that the time-axis is divided into a series of time-slots, and actions are chosen at the start of a time-slot. There are five active states, *(1)* ***Q****uiesced, (2)* ***S****tartup, (3)* ***R****unning, (4)* ***C****heckpointing, (5)* ***M****igrating and (6)* ***F****inish*, that are indexed by a number, $x(t) \in [1, 6]$. As is shown in the figure, the available actions at any given state is a subset of A, thus denoted by $A(x(t))$.

As is shown in Fig. 3, a job starts in the quiesced state. This state is also entered if under-bidding, out-of-bid events or strategic waiting (for the spot price to drop) occurs. The job is in the startup state while it waits for a VM to be provisioned. This is used to model VM startup cost. A job that is being processed is in the running state. When a job enters the checkpointing state from the running state, it stays there while the checkpoint is completed. Note that both the running and checkpointing states can be interrupted by an out-of-bid event. In the migration state, the job transfers data to a new machine and prepares to resume from that point. This state captures both the migration time and data-transfer cost. Lastly, the finish state is reached upon the job's completion. A job is assumed to remain in a state for at least one time-slot. Further, the running and checkpointing states incur VM rental costs, including partial hours, that are tracked by a parameter Δt (illustrated in the next section).

IV. Mathematical Model

The spot price of each class of spot VM fluctuates along the time-axis, forming a time-series which is described as a *spot price stochastic process* (*SPSP*). Moreover, since different classes of spot VMs are priced independently [20], each class is modeled using an independent SPSP. Markov chain has been exploited in existing literature to model the spot prices [8], [22]. Similarly, we model each SPSP as a Markov chain P_t^i, where i refers to a class of spot VMs, and t represents the time:

$$\begin{cases} Pr\left\{P_t^i = a \middle| P_{t-1}^i = a\right\} = (1 - \alpha_i) \\ Pr\left\{P_t^i = b \middle| P_{t-1}^i = a, P_t^i \neq a\right\} = f_P^i\left(b | P \neq a\right) \end{cases} \tag{1}$$

where a and b refer to two spot prices, α_i captures the rate of price fluctuations, and $f_P^i(\cdot | P \neq a)$ is a conditional probability distribution that describes the price transition pattern [8].

Given a job tuple $J(C, D, T_{dl})$, the resource agent needs to determine the policy to complete this job by the deadline. The *status* of a job at t is characterized as a tuple

[3] $\epsilon(c)$ models the checkpointing time as a function of the current job progress. For example, the model in [14], [7] can be used as $\epsilon(c)$.

[4] VMs take time to deploy and boot up. This overhead can be punitive if one switches frequently between VMs.

[5] Before checkpoining is completed, it is possible to quit checkpointing and resume at the state where checkpointing has not been started yet.

$S_t(c(t), c'(t), x(t), v(t), \Delta t))$, where $c(t)$ is the amount of remaining compute requirements (core-hours), $c'(t)$ is the amount of remaining compute requirement *since the last checkpoint* (core-hours), $x(t)$ is the job's state, $v(t)$ is the class of the VM this job is residing on, and Δt is the index of the time-slot within a complete hour. As stated in § III, this parameter tracks whether a complete VM hour is finished, and users are charged once the whole hour is reached. Note that despite the similar terminology, "status" $S_t(c(t), c'(t), x(t), v(t), \Delta t))$ and "state" $x(t)$ are two different concepts.

The life-cycle of a job corresponds to a route, connecting edges between nodes in Fig. 3. Here, we describe mathematically as to how the job's status, $S_t(c(t), c'(t), x(t), v(t), \Delta t))$, is updated with the transition between nodes (i.e., states) through edges (i.e., chosen actions) in the state transition diagram. When an out-of-bid event or a migration occurs, we update $c(t+1) = c'(t)$, which means the job's progress is lost, and it has to resume from the last checkpoint. On the other hand, when a checkpoint is completed successfully, we update $c'(t+1) = c(t)$, which implies that if an out-of-bid event occurs later, the job can resume from this new checkpoint. When the action *pr* is chosen, the job progress $c(t)$ is updated as

$$\begin{aligned} c(t+1) &= \left(1 - I_{\{out-of-bid\}}\right) \times \left[c\left(t\right) - \rho\left(v\left(t\right)\right)\right] \\ &+ I_{\{out-of-bid\}} \times c'\left(t\right) \end{aligned}$$

where $I_{\{\cdot\}}$ is an *indicator function*, which is equal to 1 if the condition holds, and $\rho(v(t))$ gives the number of cores in the VM $v(t)$ at time t.

Given the mathematical definition of the status update, the objective function can be formulated. Due to the deadline constraint, we impose the constraint $c(T_{dl}) = 0$. A policy $\pi_{(T_{dl}-1)}$, a sequence of actions $\{a_t\}$, where $t = 0, ..., (T_{dl} - 1)$, is chosen such that:

$$W_{(T_{dl}-1)} = \min_{\pi_{(T_{dl}-1)}} \mathbb{E}\left[\sum_{t=0}^{T_{dl}-1} u\left(a_t, S_{(t+1)}\right) \middle| S_0 ... S_t, \mathbf{P}_0 ... \mathbf{P}_t\right]$$

$$\text{subject to} \quad c(T_{dl}) = 0 \qquad (2)$$

where $W_{(T_{dl}-1)}$ is the expected cost spent from $t = 0$ to $t = T_{dl}$, resulting from the policy $\pi_{(T_{dl}-1)}$. The operator $\mathbb{E}$ refers to the expectation over the spot prices, S_t and $\mathbf{P}_t$ are the job's status and spot prices at t, respectively, a_t is the action taken at t, and $u\left(a_t, S_{(t+1)}\right)$ denotes the cost function of that action and the resulting status $S_{(t+1)}$. Note that the resulting status depends not only on the taken action a_t but also on the (probabilistic) spot prices $\mathbf{P}_{(t+1)}$, over which the expectation is performed.

V. Optimal Solution using RTBA

In this section, the optimal solution for Eq. 2 is found with the formalism of Bellman equation, and the solution then inspires our Reusable Table-Based Algorithm (RTBA) to effectively plan the policy.

A. Optimal Decision-making

Bellman's equation represents the value of an objective function at a decision point in terms of the sum of an action's cost and the value of the residual objective function that results from that action [5]. Given the Markov chain assumption Eq. (1) along with the *law of iterated expectations*, Eq. (2) is revised as:

$$\begin{aligned} W_{(T_{dl}-1)} &= \min_{\pi_{(T_{dl}-1)}} \mathbb{E}\left[\sum_{t=0}^{T_{dl}-1} u\left(a_t, S_{(t+1)}\right) \middle| S_t, \mathbf{P}_t\right] \\ &= \min_{a_0} \mathbb{E}\left[u\left(a_0, S_1\right) | s_0, \mathbf{P}_0\right] + W_{(T_{dl}-2)} \quad (3) \end{aligned}$$

We arrive at Bellman's equation with finite horizon from $t = 0, \ldots, (T_{dl} - 1)$. This equation reduces the choice of a policy $\pi_{(T_{dl}-1)}$ to a sequence of choices of a_t. Note that the subscript of $W_{(T_{dl}-1)}$ should not be interpreted as time. Instead, it refers to the length of the policy (i.e., the number of actions), which leads to the cost $W_{(T_{dl}-1)}$. In particular, given the current job state at time t as $x(t)$, we choose, from the set of applicable actions $A\left(x\left(t\right)\right)$ the action and its corresponding sub-policy whose sum is the minimum. Formally,

$$a_t = \arg\min_{a_t} \mathbb{E}\left[u\left(a_t, S_{(t+1)}\right) \middle| s_t, \mathbf{P}_t\right] + W_{(T_{dl}-t-2)} \qquad (4)$$

which suggests a recursive relationship for the optimal decision-making routine.

B. Action Profile

We illustrate in Fig. 4 how the resource agent chooses the actions in response to spot price fluctuations, termed as *action profiles*. The test period of the spot prices is chosen from 00:00 a.m. to 10:00 a.m. on Dec, 19, 2012, when the medium spot VM (dotted blue line) has more fluctuations in the first half of the observation period while the large spot VM (dashed green line) has more fluctuations in the second half. Further, the SPSP model is based on the historic spot prices from Aug, 2012 to Oct, 2012, the job's compute requirement (C) is set to 8 core-hours, and the deadline (T_{dl}) is chosen as 10 hours.

In Fig. 4, the chosen actions for the job are represented as a solid orange line, and the out-of-bid events and checkpointing are identified with ✗ and ▲ signs, respectively. In the realistic scenario, VMs are initiated after a startup delay; this is visible in the brief time spent in the *Startup* action prior to the job *Running* on a VM. There are three out-of-bid events during the job's life-cycle – the first two occur while the VM is starting and the third one revokes the large spot VM from the resource agent, causing the progress made after the second checkpoint to be lost.

Since the large spot VMs are cheap at the beginning, the resource agent selects them initially to run the job.

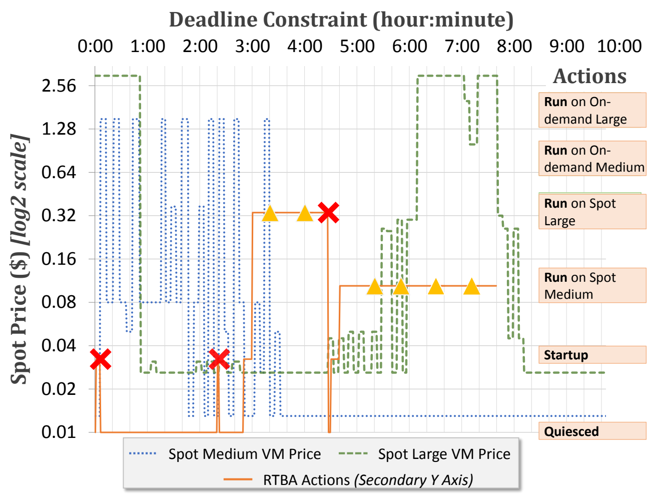


Figure 4: The X-axis shows the job's life time. The primary Y-axis shows the current spot prices of medium and large VMs (dotted blue and dashed green lines respectively). The secondary Y-axis shows various actions taken by the RTBA algorithm (solid orange line). The ✗ and ▲ signs represent out-of-bid events and checkpointing.

The resource agent bids a large VM at 2:50 a.m. and then performs checkpoints of its progress. After an out-of-bid event strikes the large VM at 4:27 a.m., the resource agent has to re-bid for a new VM and resume the job from the previous checkpoint. Interestingly, this time the resource agent selects a medium spot VM since it predicts that the price of large VMs is likely to increase (as it does in the latter half of the test period). The more stable-priced medium spot VMs are used to finish the job at 7:40 a.m, with period checkpoints and well ahead of the deadline.

C. *Reusable Table-Based Algorithm (RTBA)*

As is shown in Eq. 4, the optimal action at any time can be determined if W_τ is evaluated in a *backward* fashion from the base case $\tau = 0$. Note that the subscript of W_τ refers to the length of the policy, which leads to the cost W_τ. W_0 represents the base case since it corresponds to the deadline, when no more actions need to be decided. Intuitively, the base case falls into one of two situations: completed jobs ($c(T_{dl}) = 0$) and incomplete jobs ($c(T_{dl}) > 0$). Since the latter case fails the deadline constraint, we set its $W_0 = \infty$. On the other hand, the former implies the job has finished at or prior to the deadline. Hence, $W_0 = 0$ if $\Delta t = 0$; $W_0 = p_t^{v(0)}$ if $\Delta t \neq 0$. This is because if a new VM-hour cycle has not yet started ($\Delta t = 0$), users are not charged. Else, users are charged according to the VM's price, $p_t^{v(0)}$.

Starting from the base case, one can obtain the optimal policy by solving Eq. 4 in a backward fashion, from $t = (T_{dl} - 1)$ to $t = 0$, for every combination of $(c(t), c'(t))$. We solve this using a *dynamic programming* framework, which stores the value of the objective function at $t = t'$

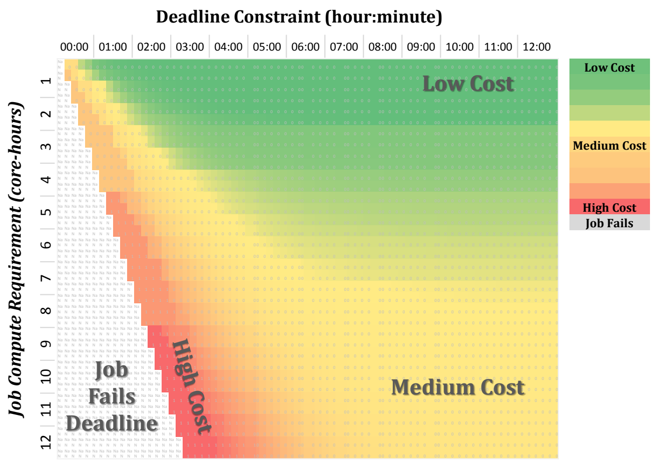


Figure 5: Heatmap showing expected cost of running jobs with different deadline constraints (X-axis, hours:mins) and compute requirements (Y-axis, core-hours). The costs are described by a color scale, green being cheap and red costly. The grey region in the bottom left indicates jobs with short deadlines that fail to meet the constraint.

for the evaluation of the value at $t = (t' - 1)$. Thus, the complexity is $O(C^2 T_{dl})$.

Scanning and plotting the solution space as a *heatmap* in Fig. 5 offers visual insight. We build the SPSP model using historic spot price data from Aug, 2012 to Oct, 2012, and programmatically solve the optimization problem for diverse combinations of compute requirements and deadlines. The value of the objective function for different deadlines (X-axis) and compute requirements (Y-axis) is depicted using a color scale. This heatmap highlights the optimal trade-offs between compute requirements, deadlines and budgets, and helps answer the following questions: (1) *Given a budget and the job's compute requirement, what is the minimum deadline can one set?* (2) *Given the job's compute requirement and deadline, how much would one expect to spend?* and (3) *Given a job's compute requirement, where is the "sweet spot", where a small concession on the deadline offers a significant cost reduction (i.e., sharp color-transitions in the heatmap)?*

This solution inspires our Reusable Table-Based Algorithm (RTBA). Since a table is being constructed when we solve the optimization problem backwards, this table, termed as the *strategy table*, can be queried whenever an action needs to be decided. Notably, the strategy table is reusable for other jobs if their compute requirement C and deadline constraint T_{dl} fall within this table's bound. This is because if the job falls within this bound, there should be an entry in the strategy table which corresponds to this job, and which was solved as a sub-problem when constructing the table using dynamic programming. Essentially, RTBA constructs a strategy table offline, and when an action is to be decided, performs a simple table look-up. The table can be reused and expanded if larger bounds are necessary. Importantly,

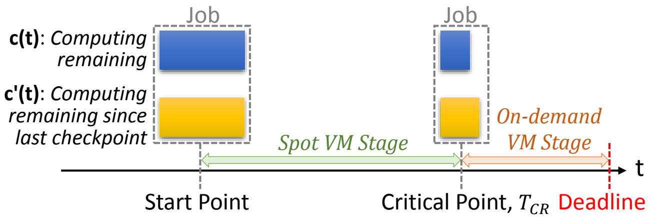


Figure 6: The value of residual compute requirement, $c(t)$, is represented on top as a blue block, and the residual compute required since the last checkpoint, $c'(t)$, is represented below as a yellow block. At the start, $c'(t) = c(t)$ since no checkpoints have been made. For H1, $c'(T_{CR}) = c(0)$, and for H2, $c'(T_{CR}) = c(T_{CR})$, where T_{CR} remains constant. For H3, $c'(T_{CR}) = c(T_{CR})$, but T_{CR} is pushed forward.

while the construction of the strategy table depends on the modeled SPSP, whose pattern might change over time [2], we experimentally observe that RTBA is robust over time due to the probabilistic approach to spot price modeling [23]. Therefore, near-optimal solutions can be achieved even without frequently reconstructing the strategy table to include recent spot prices.

VI. Heuristic-based Solutions

While RTBA offers an optimal solution, constructing the strategy table can be costly, particularly if the number of jobs is too small to amortize the construction cost, or the job diversity is too large to enable reuse. We propose three heuristics to facilitate agile decision-making for *ad hoc* jobs. These heuristics also offer baselines of increasing sophistication to compare against RTBA.

Spot prices are generally much cheaper than their on-demand counterparts. Intuitively, the more one uses spot VMs, the less they are going to spend overall. Essentially, our heuristics process a job using spot VMs until a potential out-of-bid event would threaten the job's feasibility to complete on-time. We define *critical point*, T_{CR}, as the time beyond which a job cannot be completed after resuming from its checkpoint. This illustrated in Fig. 6.

$$T_{CR} = \frac{c'(t)}{\rho\left(v\left(t\right)\right)} \tag{5}$$

Thus, the critical point is a function of $c'(t)$, the job's remaining compute requirement at the last checkpoint, and $\rho\left(v\left(t\right)\right)$,the processing capability of its current (spot) VM. The critical point locates the transition from spot to on-demand VMs. We refer to the time prior to the critical point as *spot stage* and to that after the critical point as *on-demand stage*. The three heuristics vary in when they perform checkpoints. We define the *naïve critical point* by setting $c'(t) = C$ in Eq. 5. The naïve critical point is conservative in choosing the transition point, and ensures that the job can *restart* and complete even if no checkpoints have been performed in the spot stage.

H1:Two-Stage Algorithm. This heuristic uses only the spot VMs prior to the naïve critical point, does not perform any checkpoints. It *restarts* the job on the on-demand VMs if the job is not completed by the naïve critical point. This can lead to over-payments since the expenditure on the spot stage does not contribute to the job's completion if it did not finish before the naïve critical point.

H2:Boundary Checkpoint Algorithm. This heuristic is similar to H1 but attempts to transfer the progress made in the spot stage to the on-demand stage. When the naïve critical point is reached, this heuristic performs a checkpoint prior to switching to on-demand VMs, and resumes the job from that checkpoint. However, it is still vulnerable to out-of-bid events close to the naïve critical point.

H3:Rolling Checkpoint Algorithm. This heuristic tries to push forward the critical point to allow the job to continue running on spot VMs for longer. Due to the functional relationship between T_{CR} and $c'(t)$, we periodically checkpoint and update $c'(t)$. We define the *effective frequency*, F_{eff}, as the duration between two consecutive checkpoints. Ideally, F_{eff} should allow the job to progress sufficiently to outweigh the cost of checkpointing. For simplicity, we fix F_{eff} to be equal to the average lifetime of a particular spot CM class multiplied by its number of cores. A more sophisticated approach can set F_{eff} as a function of the current spot price.

VII. Results

A. Simulation Setup

We consider five machine types, $\langle$*Spot medium*, *Spot large*, *On-demand medium*, *On-demand large*, *Local medium*$\rangle$. The medium machines have 2 CPU cores while the large VMs have 4 cores. We use Amazon EC2's pricing policy on *us-east-1a* data center [6]. Each time-slot t in our model spans 10 minutes. The overheads for VM startup, checkpointing and migration between the cloud and local machine occupy one time-slot. On the other hand, the overhead for migration within the cloud needs zero time-slots, assuming it can interleave with VM startup time. For the job $J(C, D, T_{dl})$, we vary the compute requirement C between $2 - 8$ core-hours while the input data size and deadline constraint are also varied in 10 minute increments. Our focus is on *long-running* jobs (i.e., $O(hours)$) which cumulatively cost more and hence have more to gain.

The strategy table of RTBA algorithm is constructed on a server with 2x8-core AMD Opteron 3.0GHz CPU and 64GB RAM. The time to construct the strategy table for different C and T_{dl} values is measured. However, due to the limited space, we report the time to build the table for an 8 core-hour job with a 12 hour deadline as 24.74 minutes on average. For our SPSP model, $f_P^i\left(\cdot|P \neq a\right)$ can be obtained through

[6]Medium VM-hour costs \$0.13, Large VM-hour costs \$0.26, and the data transfer-in and -ot costs are \$0.00 and \$0.12 per GB

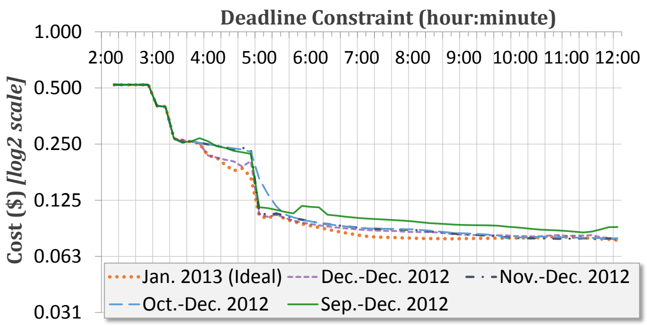


Figure 7: Average cost (Y-axis) to run 8 core-hour job in Jan, 2013 with various deadlines (X-axis) using RTBA. Plots show different training data used in the SPSP model. Using older data reduces cost benefits. Jan, 2013 line is ideal case.

the histogram of the spot price history. Since our SPSP model implies that the interval between two consecutive price changes forms a *geometric distribution* with parameter α_i, we set α_i as the inverse of the empirical mean of the intervals between two consecutive price changes [12]. We discretely choose the four most probable spot prices as our bidding prices, incremented by $0.001 as a safety buffer.

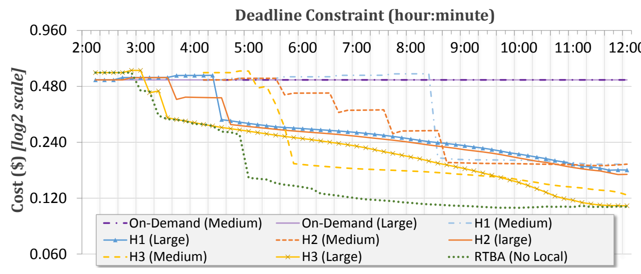
B. *Impact of Training Interval*

In the first experiment, the impact on cost reductions by using different training intervals to build the SPSP cost model is investigated. Four training interval durations in 2012, $\langle$Sep-Dec, Oct-Dec, Nov-Dec, Dec-Dec$\rangle$, are used to build four SPSP models, which are then used by the resource agent to construct a strategy table for scheduling the 8 core-hour job with varied deadlines in Jan, 2013. Further, we also build an SPSP model using Jan, 2013 data *post facto* as an ideal case. Fig. 7 reports the average of costs obtained for running the 8 core-hour job for each deadline, once for every 1 minute sliding window in Jan, 2013, i.e. each data point is averaged on $31 \times 24 \times 60$ runs.

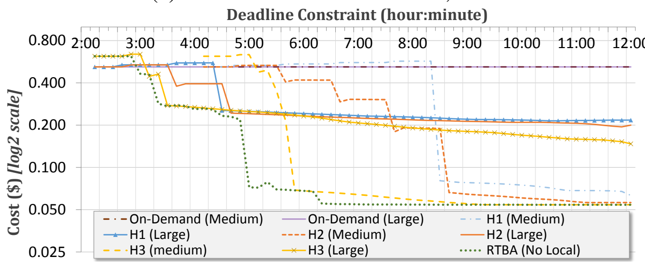
When the training data is more outdated, the cost reduction becomes less. For e.g., in Fig. 7, *Dec-Dec, 2012* appears closest to the ideal case of *Jan, 2013*. This observation agrees with [2], which indicates that Amazon's pricing mechanism changes between *epochs*. Notably, while the SPSP model trained with older data digresses from the contemporary price pattern, the performance of RTBA only degrades slightly, demonstrating the robustness of our approach. So, it is possible to reuse, rather than often reconstruct, the strategy table even with varying price patterns.

C. *Performance Comparison between Algorithms*

In the second experiment, the performances of RTBA, H1, H2, and H3 are compared. Importantly, we investigate the influence of diverse job deadlines on the job's cost. As is intuitive, more relaxed deadlines offer more opportunities for the resource agent to help reduce costs. We use 8 core-hours as our candidate job's compute requirement.



(a) Simulation results for Nov, 2012



(b) Simulation results for Dec, 2012

Figure 8: Average cost (Y-axis) to run 8 core-hour job with different deadlines (X-axis) using proposed strategies, in (a) Nov, 2012 and (b) Dec, 2012. The cost when using only on-demand medium or large VMs remains flat at $0.52.

We use spot prices from Aug-Oct 2012 as training data for the SPSP model. Two sets of simulations are conducted – for jobs submitted in Nov, 2012 and in Dec, 2012. The spot price patterns for these two months are very different (Fig. 1). The spot prices in Nov, 2012 have standard deviations of σ_{medium} = $4.88 and σ_{large} = $0.1419 for medium and large VMs. Nov is more variable than Dec, 2012 prices, which have σ_{medium} = $0.5284 and σ_{large} = $0.3766 for the respective VMs. In Fig. 8, we plot the average cost for completing the 8 core-hour job on the Y-axis with different deadlines on X-axis when using only on-demand VMs, or the RTBA, H1, H2, or H3 strategies. As before, we repeat the job for every 1 minute interval in Nov, 2012 (Fig. 8a) and Dec, 2012 (Fig. 8a), and report the average costs. Note that the on-demand approach and the heuristics operate on a single VM class during a job's life-cycle. While RTBA can switch between hybrid resources, we disable the access to local machine for a fairer, cloud-centric comparison. The heuristics use a bid price set at the most probable spot price, incremented by an additional $0.001 to offer a bid advantage. For H3, the effective frequency is statically set as the average lifetime of the considered VM class multiplied by its computing power (i.e., number of cores).

In general, except for the on-demand only approach, the costs reduce as the deadlines are relaxed for all our algorithms. RTBA is best able to exploit the looseness of the deadline and outperforms all others (green dotted line on the left), providing the minimum (optimal) costs. Further, we observe a few sharp drops, termed as *cost cliffs*, for our proposed strategies. These result from Amazon's policy of

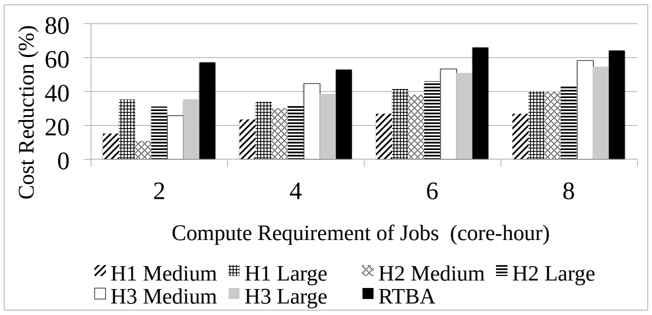


Figure 9: The cost reductions for the four algorithms relative to using only on-demand VMs are shown for Nov, 2012. The heuristics H1–H3 only operate on a unique VM class while RTBA can switch VM classes. Hence, two columns (medium and large VM) are shown for each heuristic.

charging in hourly VM increments. Note that these cliffs are also visible as sharp color transitions in the heatmap (Fig. 5), that offers a parameter sweep of jobs and deadlines.

In RTBA, the cost cliffs appear earlier and are deeper than for H3, and similarly with H3 and H2, and with H2 and H1. More sophisticated approaches are more capable of advanced planning and checkpointing of the job's progress. The cliffs are inflection points. Deadlines that are in the proximity of these cost cliffs can benefit (or suffer) from small relaxations (tightening) of the deadlines it offer significant cost reductions (penalties). On the other hand, the heuristics also exhibit cases where they *overpay*, relative to the on-demand only approach. These result from the progress lost by the spot VMs due to out-of-bid events. In particular, H1 is the most vulnerable due to the absence of checkpointing while the overpayment is alleviated in H2 and H3. Notably, the performance of H3 is close to that of RTBA when the deadline constraint is loose, and the spot prices are stable, as shown at the tail of the curves in Fig. 8b.

D. *Impact of Compute Requirement*

In the third experiment, we use the same configuration as the second one except for varying the compute requirement for the job. The cost reductions achieved for our algorithms, normalized against using only on-demand VMs, are compared for a range of compute sizes (2, 4, 6, 8 core-hours), when run in Nov, 2012. The cost reduction for each compute size is averaged over a range of deadline constraints, from $1\times$ to $3\times$ of the compute size. As can be expected, Fig. 9 shows that the relative cost reduction of RTBA outperforms the other heuristics. However, as the compute requirement becomes larger, the advantage of RTBA shrinks. Further, the overhead to calculate the RTBA strategy table becomes acute for job's with large compute sizes. Given these, the RTBA and H3 algorithms are complementary; RTBA is advantageous for small sized or repetitive jobs while H3 is more useful for *ad hoc* job workloads or large jobs with punitive costs for building the strategy table.

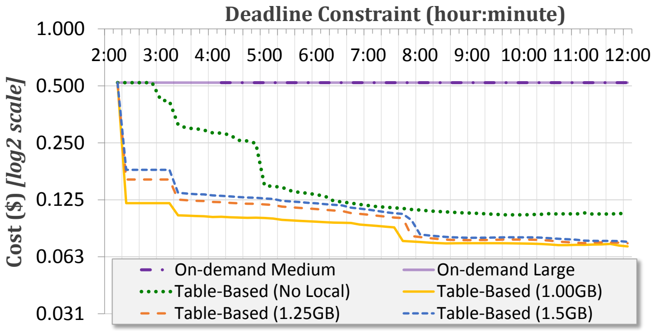


Figure 10: The costs (Y-axis) for processing 8 core-hour jobs with varied input data sizes. RTBA with and without the local machine converge as the deadline increases.

E. *Impact of Input Data Size*

We investigate the relationship between a job's input data size D and cost to run the job. We retain the 8 core-hour job with different deadlines, and run it for Nov, 2012, averaging the costs from over a 1-minute sliding window. We compare RTBA with the on-demand only VMs, and further compare RTBA with and without access to local machines. Since the local machine is "free", there is an inclination to switch to them. However, as the job and its input data are initially in the cloud, there are bandwidth charges proportional to D that are incurred to ship the job to the local machine.

As is shown in Fig. 10, for the tightest deadline constraint, the four curves for the RTBA algorithms cost the same as using only on-demand VMs since there is not enough time to migrate out of the cloud or to use spot VMs. But, as the deadlines are relaxed, the difference between RTBA with and without the local machine narrows as the cost of getting the work done on cheaper spot VMs approaches the data transfer cost to the free, but scarce, local machine.

VIII. Conclusion and Future Work

Our work addresses the issue of effective and economic use of hybrid clouds for managing the life-cycle of long running jobs. It actively leverages the cheaply-priced spot VMs that have been less studied so far. A resource agent serves as a "personal" adviser for individual jobs running on exclusive VMs, with the goal of suggesting the right actions to meet the specified deadlines while minimizing rental costs. Our system and job models attempt to mimic real-world clouds and applications. Our simulation study is based on observed spot and on-demand VM prices on Amazon EC2, and incorporates both compute costs and asymmetric network bandwidth rates charged by cloud providers.

While the computational complexity of RTBA makes it tractable only for repetitive jobs, the H3 heuristic is simpler while nearing optimality in empirical studies. Hence, our work goes beyond a theoretical stochastic optimization problem and can be used in practice for average cost savings of $60\% - 72\%$ for H3 and RTBA.

Our approach can be extended along two dimensions. RTBA can be modified to accommodate the scenario where jobs are allowed to spillover the deadline but incur some penalty. The resource agent tends to be more aggressive when the penalty is outweighed by the saving which results from applying speculative strategies. Secondly, the QoS can be defined in terms of the rate of successful job completions, allowing the agent to balance risks and rewards.

The agent's role can be viewed from another perspective. As a *resource broker*, the agent owns some local resources to process end-user requests. By monitoring the spot market, the agent can decide whether to use its own resources or cheaper spot VMs to process jobs so that fragmented computing resources can be leveraged for productive work. This, in the spirit of the spot market, is a win-win-win situation for the broker, end-users and the cloud provider.

Acknowledgment

The authors would like to thank Viktor Prasanna, Alok Kumbhare, and Charith Wickramaarachchi of USC for their feedback. This research is supported by DARPA's XDATA program and National Science Foundation under award Nos. CCF-1048311 and ACI-1216898.

References

[1] Amazon Spot Market. aws.amazon.com/ec2/spot-instances.

[2] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2011.

[3] A. Andrzejak, D. Kondo, and S. Yi. Decision Model for Cloud Computing under SLA Constraints. In *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, Aug. 2010.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *ACM Communications*, 53(4):50–58, Apr. 2010.

[5] D. P. Bertsekas. Dynamic Programming and Optimal Control. *Athena Scientific*, I, 2005.

[6] T. Bicer, D. Chiu, and G. Agrawal. Time and Cost Sensitive Data-Intensive Computing on Hybrid Clouds. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2012.

[7] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing Strategies for Parallel Jobs. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011.

[8] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See Spot Run: Using Spot Instances for Mapreduce Workflows. In *USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, Jun. 2010.

[9] M. D. de Assuncao, A. di Costanzo, and R. Buyya. Evaluating the Cost-benefit of Using Cloud Computing to Extend the Capacity of Clusters. In *ACM International Symposium on High Performance Distributed Computing (HPDC)*, Jun. 2009.

[10] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The Cost of Doing Science on the Cloud: the Montage Example. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2008.

[11] S. Di and C.-L. Wang. Error-Tolerant Resource Allocation and Payment Minimization for Cloud System. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1097–1106, Jun. 2013.

[12] G. Grimmett and D. Welsh. Probability: An Introduction. *Oxford Science Publications*, 1986.

[13] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Data Sharing Options for Scientific Workflows on Amazon EC2. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2010.

[14] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. PaÃÜun, and S. Scott. An Optimal Checkpoint/Restart Model for A Large Scale High Performance Computing System. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2008.

[15] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski. Cost- and Deadline-constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2012.

[16] M. Mao and M. Humphrey. Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[17] P. Marshall, K. Keahey, and T. Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. In *IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010.

[18] M. Mazzucco and M. Dumas. Achieving Performance and Availability Guarantees with Spot Instances. In *IEEE International Conference on High Performance Computing and Communications (HPCC)*, Sep. 2011.

[19] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, Modeling, and Evaluation of A Scalable Multi-level Checkpointing System. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2010.

[20] W. Voorsluys and R. Buyya. Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Mar. 2012.

[21] S. Yi, D. Kondo, and A. Andrzejak. Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In *IEEE International Conference on Cloud Computing (CLOUD)*, Jul. 2010.

[22] M. Zafer, Y. Song, and K.-W. Lee. Optimal Bids for Spot VMs in A Cloud for Deadline Constrained Jobs. In *IEEE International Conference on Cloud Computing (CLOUD)*, Jun. 2012.

[23] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang. Optimal Resource Rental Planning for Elastic Applications in Cloud Market. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2012.

[24] L. Zhao, Y. Ren, and K. Sakurai. A Resource Minimizing Scheduling Algorithm with Ensuring the Deadline and Reliability in Heterogeneous Systems. In *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Mar. 2011.