2012 26th IEEE International Conference on Advanced Information Networking and Applications

Reliable Provisioning of Spot Instances for Compute-intensive Applications

William Voorsluys and Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email:{williamv,rbuyya}@csse.unimelb.edu.au

***Abstract*—Cloud computing providers are now offering their unused resources for leasing in the spot market, which has been considered the first step towards a full-fledged market economy for computational resources. Spot instances are virtual machines (VMs) available at lower prices than their standard on-demand counterparts. These VMs will run for as long as the current price is lower than the maximum bid price users are willing to pay per hour. Spot instances have been increasingly used for executing compute-intensive applications. In spite of an apparent economical advantage, due to an intermittent nature of biddable resources, application execution times may be prolonged or they may not finish at all. This paper proposes a resource allocation strategy that addresses the problem of running compute-intensive jobs on a pool of intermittent virtual machines, while also aiming to run applications in a fast and economical way. To mitigate potential unavailability periods, a multifaceted fault-aware resource provisioning policy is proposed. Our solution employs price and runtime estimation mechanisms, as well as three fault-tolerance techniques, namely checkpointing, task duplication and migration. We evaluate our strategies using trace-driven simulations, which take as input real price variation traces, as well as an application trace from the Parallel Workload Archive. Our results demonstrate the effectiveness of executing applications on spot instances, respecting QoS constraints, despite occasional failures.**

***Index Terms*—cloud computing; spot market; scheduling; fault-tolerance;**

I. Introduction

Variable pricing virtual machines (also know as "spot instances"[1]) are increasingly being employed as a means of accomplishing various computational tasks, which are common in several areas of science, such as climate modeling, drug design, and protein analysis, as well in data analytics scenarios, such as execution of MapReduce tasks [1]. Significant cost savings and the possibility of easily leasing extra resources when needed, are major considerations when choosing virtual clusters, dynamically assembled out of cloud computing resources, over a local HPC cluster [2].

The cloud computing spot market, since introduced by Amazon Web Services [3], [4], has been considered as the first step for a full-fledged market economy for computational resources [5]. In this market, users submit a resource leasing request that specifies a maximum price (bid) they are willing to pay per hour for a predefined instance type. Instances associated to that request will run for as long as the current spot price is lower than the specified bid. Prices vary frequently, based on supply and demand. Price are distinct and vary independently for each available datacenter ("availability zone" in Amazon terminology), spot instance type, and operating system choice. Not all type/OS combinations are available in all datacenters. In other words, there are multiple spot markets from where to choose suitable computational resources, making the provisioning problem significantly challenging.

When an out-of-bid situation occurs, i.e. the current spot price for that instance type goes above the user's maximum bid, instances are terminated by the provider without prior notice. Therefore, in spite of an apparent economical advantage, an intermittent nature is inherent to biddable resources, which may cause VM unavailability.

Despite the possibility of failures due to out-of-bid situations, as we have discussed in our previous work [2], it is advantageous to utilize spot instances to run compute-intensive applications at a fraction of the price that would normally cost when using standard fixed-priced VMs. Specifically, we have demonstrated the effect of different runtime estimation methods on the decision-making process of a dynamic job allocation policy. Our policy was responsible for requesting and terminating spot instances on-the-fly as needed by a stream of computational jobs, as well as choosing the best instance type for each job based on the estimated job execution time on each available type.

We had previously assumed that users would bid high enough so that the chance of spot instance failures due to out-of-bid situations would be negligible. In reality, even though users only pay the current spot price at the beginning of each hour, regardless of the specified bid, there are incentives for bidding lower. Andrzejak et al, who evaluated checkpointing techniques for spot instance fault-tolerance, observed that by bidding low, significant cost savings can be achieved, but execution times increase significantly. Similarly, by increasing the budget slightly, execution times can be reduced by a large factor [6].

A. Bidding strategies and the need for fault-tolerance

We now elaborate on the potential risks and rewards of provisioning a resource pool composed exclusively of spot

[1]The terms "spot instance", "instance", "virtual machine", "VM", and "resource" signify the same concept and are used interchangeably in this work.

1550-445X/12 $26.00 © 2012 IEEE
DOI 10.1109/AINA.2012.106

instances in scenarios where QoS constraints play an import role.

Failures due to out-of-bid situations may lead to the inability to provide the desired quality of service, e.g.: prolonged application execution times or an inability of applications to finish within a specified deadline. To overcome this uncertainty, one may come up with a few strategies to decrease the chance of failure or mitigate their effects.

To decrease the chance that out-of-bid situations occur, one could to choose to bid as high as possible. Given that, under the current model of Amazon spot instances, users pay at maximum the current spot price (not the actual bid), there would be no apparent disadvantages in bidding much higher than the spot price. However, there are incentives for adopting more aggressive bidding strategies, i.e. bidding close or even lower than the current spot price.

Firstly, Amazon offers on-demand instances at a fixed price, which are identically functional to spot instances and are not subject to terminations due to pricing issues. The value set by Amazon to these on-demand instances is likely to influence the maximum price a user is willing to bid. Thus, this value acts as an upper bound for bids of users that would rather lease a more reliable on-demand instance in cases the spot price is equal or above the on-demand price. In fact, by analysing the history of spot prices of Amazon EC2, we have observed that, over the period of about 100 days from 05-Jul-2011 to 15-Oct-2011, spot prices have surpassed on-demand prices several times across most instances types and datacenters. For example, the spot price of one of the most economical instances (M1SMALL) in the US-EAST region, has reached this situation 11 times, for periods of up to 2 hours and 20 minutes, and price value of up to 17% above the on-demand price.

Secondly, in a scenario where most users submit high bids, providers would likely increase the spot price to maximize profits. As previously postulated [7], the Amazon EC2 spot market resembles a Vickrey auction style [8], where users submit sealed bids, the provider gathers them and computes a clearing price. The pricing scheme thought to be used by Amazon, where all buyers pay the clearing price, is a generalization of the Vickrey model for multiple divisible goods, the standard uniform price auction, on which the provider assigns resources to users starting by the highest bidder, until all bids are satisfied or there are no more resources. The price paid by all users is the value of the lowest winning bid (sometimes, the highest non winning bid) [5]. It has also been observed that Amazon may be artificially intervening in the prices by setting a reserve price and generating prices at random [9]. In any case, we argue that there is an incentive for users to submit fair bids, based on the true value they are willing to pay for the resource.

Thirdly, on a similar note, users may choose to postpone non-urgent tasks when prices are relatively high, hoping to obtain a lower price (the true value) later, a strategy that can be accomplished by placing a bid at the desired price and waiting for it to be fulfilled. Similarly, in the case of an out-of-bid situation, owners of a non-urgent task would prefer wait for the request to be in-bid again, rather than obtaining a new resource under new lease terms (e.g. another VM type, or the same type at a higher bid).

Finally, as observed by Yi et al [10], one can bid low to take advantage of the fact that the provider does not charge the partial hour that precedes an out-of-bid situation. Thus, delaying the termination of an instance, even when it is not needed, to the next hour boundary, one can expect a probability of failure before termination, potentially avoiding to pay for the last hour.

The choice of an exact bid value can be empirically derived from a number of factors, including observations of price history, the willingness of the user to run instances at less than a certain price or not run at all, and a minimum reliability level required. These factors, when reflected on the bid value, define how likely the system is able to meet time and cost constraints.

In any case, the adoption of more aggressive bidding strategies can result in more failures, and potentially undermine the cost savings, as a result of frequent loss of work. Therefore, resource provisioning policies aimed at running computational jobs on spot instances must be accompanied by fault mitigation techniques, especially tailored for the features of cloud computing spot instances. Notable features of spot instances may influence the way fault-tolerance works in this scenario. Most notably, an hour-based billing granularity and non-payment of partial hours in the case of failures, guarantees payment of the actual progress of computation [10]. Additionally, given that providers, such as Amazon, freely provide a history of price variations, significantly more informed decisions can be made by observing the past behaviour.

B. Our contribution

This paper proposes a resource provisioning strategy that addresses the problem of running computational jobs on intermittent spot instances. In particular, we aim to perform this in a reliable manner, similar to what would be achieved if on-demand instances were chosen instead. Our main objective is to run applications in a fast and economical way, while tolerating sudden unavailability of virtual machines. We build up on our previous work [2], where we demonstrated the viability of dynamically assembling virtual clusters exclusively composed of spot instances to run compute-intensive applications.

Specifically, the contributions of this work are:

- A multifaceted resource provisioning approach, that includes novel mechanisms for maximizing the chance jobs finish within their deadlines, while minimizing costs in a spot instances-based computational platform;
- A bidding mechanism that aids the decision-making process by estimating future spot prices and making informed bidding decisions;
- An evaluation of two novel fault-tolerance techniques, namely migration and job duplication, and their comparison to an existing checkpointing-based approach, in terms of deadline violations and cost.

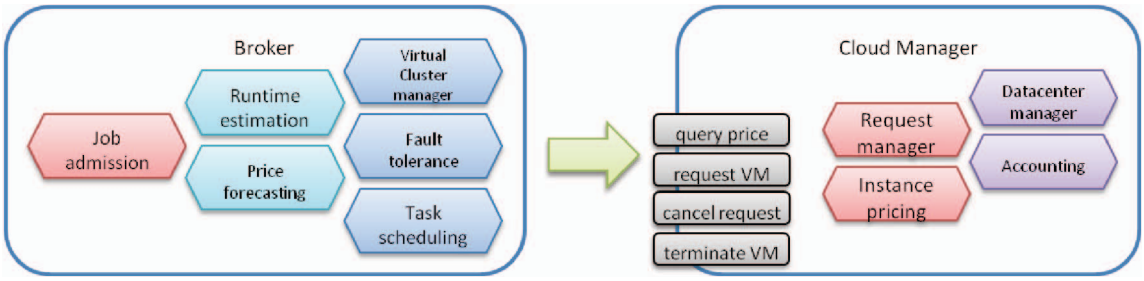


Fig. 1. Modeled architecture: Client (broker) and server (cloud) side. The "Runtime estimation" component was the focus of our previous work [2]. Here, we focus primarily on the "fault-tolerance" component

The rest of this paper is organized as follows: Section II describes related literature on existing approaches that use spot instances; Section III describes our existing resource provisioning policy and discusses the modifications necessary to add a reliability component to it; Section IV details our multifaceted approach and discusses each mechanism and the interaction between them; Section V presents extensive simulation-based experimental results and their discussion; finally Section VI concludes the paper.

II. Related Work

A few recently published works have touched the subject of leveraging variable pricing cloud resources in high-performance computing. Andrzejak et al. [6] have proposed a probabilistic decision model to help users decide how much to bid for a certain spot instance type in order to meet a certain monetary budget or a deadline. The model suggests bid values based on the probability of failures calculated using a mean of past prices from Amazon EC2. It can then estimate, with a given confidence, values for a budget and a deadline that can be achieved if the given bid is used.

Yi et al. [10] proposed a method to reduce costs of computations and providing fault-tolerance when using EC2 spot instances. Based on the price history, they simulated how several checkpointing policies would perform when faced with out-of-bid situations. The proposed policies used two distinct techniques for deciding when to checkpoint a running program: at hour boundaries and at price rising edges. In the hour boundary scheme, checkpoints are taken periodically every hour, while in the rising edge scheme, checkpoints are taken when the spot price for a given instance type is increasing. The authors proposed combinations of the above mentioned schemes, including adaptive decisions, such as taking or skipping checkpointing at certain times. Their evaluation has shown that checkpointing schemes, in spite of the inherent overhead, can tolerate instance failures while reducing the price paid, as compared to normal on-demand instances. Similarly, we evaluate a checkpointing mechanism implemented according to this work, with the objective of comparing with other fault-tolerance approaches.

III. Resource Provisioning in a Spot Instances-based Computational Platform

In our previous work, we have proposed a resource provisioning and job allocation architecture and an associated policy. Our solution has been tailored for an organization that aims at assembling a computational platform solely based on spot instances and use it to accomplish a stream of deadline-constrained computational jobs. In that work, we also evaluated several runtime estimation mechanisms and their effect on cost and utilization of the platform, as well as deadline violations of jobs. Especially, we have employed the same workload we use in this work, and compared the costs of running such workload on on-demand and on spot instances. In this section, we summarize how our solution works; a detailed description and analysis can be found in [2].

A Broker component is responsible for receiving computational job requests from users, provisioning a suitable VM pool by interacting with the provider, and applying a job scheduling policy to ensure jobs finish within their deadlines, while minimizing the cost. A diagram depicting the components of the modeled architecture is shown in Figure 1.

We have modeled a cloud computing provider according to how Amazon EC2 currently works in practice. The provider manages a computational cloud, formed by one or more datacenters, which offer virtual machines of predefined types in a spot market. The provisioning of an instance is subject to the following characteristics: clients submit requests for a single instance, specifying a type, and up to how much they are willing to pay per instance/hour (bid). Optionally, a particular datacenter can be specified; if left blank, the provider allocates the instance to the most economical datacenter choice. The system provides instances whenever the bid is greater than the current price; on the other hand, it terminates instances without any notice when a client's bid is less than or equal to the current price. The system does not charge the last partial hour when it stops an instance, but it charges the last partial hour when the termination is initiated by the client (the price of a partial hour is considered the same as a full hour). The price of each instance/hour is the spot price at the beginning of the hour.

Jobs are assumed to be moldable, in the sense that they can run on any number of CPU cores, but limited to a single

virtual machine. To determine the run time of a job in a particular number of CPU cores, we use Downey's analytical model for job speedup [11]. To generate values for A (average parallelism) and σ (coefficient of variance of parallelism), we have used the model of Cirne & Berman [12]. The moldability of a job defines it's preferred instance type, i.e. the type on which the job will take advantage of the most number of cores for a time greater than 1 hour. As a result, longer jobs that offer more parallelism will prefer instances with more cores.

The activities of our proposed algorithm are summarized in the steps described below.

- When any job is submitted, it is inserted into a list of unscheduled jobs;
- At regular intervals (T), the algorithm uses a runtime estimation method to predict the approximate runtime of the job on each available instance type;
- The broker then attempts to allocate the job to an idle VM with enough time before a whole hour finishes;
- If unsuccessful, it attempts to allocate the job to a VM that is currently running jobs but is expected to become idle soon. Runtime estimates of all jobs running on the VM, in addition to the incoming job, are required at this step;
- If the job still cannot be allocated, the algorithm will decide whether it is advantageous to extend a current lease, to start a new VM lease, or to postpone the allocation decision according to the job's urgency factor and pricing conditions.

The urgency factor U of a job j is the maximum estimated time the job can wait for a resource to be provisioned so that the chance of meeting the deadline is increased. It is computed as per Equation 1, where D_j is the job's deadline, T is the current time, so that $D_j - T$ corresponds to the time until the job's deadline; α is the urgency modifier; e_j is the estimated runtime of j on it's preferred instance type; and B is the expected time the provider takes to provision a new VM (fixed at 5 minutes).

$$U_j = max(0, D_j - T - (\alpha * e_j + B)) \quad (1)$$

The greater the value of the α modifier, the more conservative the algorithm becomes, i.e. with higher values of α, U approximates 0. A value equal to 0 indicates that a resource must be provisioned immediately to complete the job within the deadline. Alternatively, lower values of α cause the algorithm to postpone more provisioning actions in order to maximise the chances of finding lower prices or reusing other jobs' instances.

IV. Mechanisms to Achieve Fault Tolerance

In this work, we explore a multifaceted approach, which relies on two interrelated modalities that define how reliably the policy ensures that computational jobs finish before their deadlines. The first mechanism aims at choosing appropriate bid values based on estimation of price variations and on the job's urgency factor U, which influences the choice of when to provision a resource for a given job and how much to bid. The second mechanism adds extra levels of fault-tolerance through checkpointing and migration of virtual machines, as well as job duplication.

TABLE I
Evaluated bidding strategies

Bidding strategy	Bid value definition
Minimum	The minimum value observed in the price history + G
Mean	The mean of all values in the price history
On-demand	The listed on demand price
High	A value much greater than any price observed (defined as 100)
Current	The current spot price + G

These mechanisms aim at mitigating spot instance unavailability due to out-of-bid situations only, i.e. failures due to price variations. Other types of instance failures, for instance, due to hardware faults or network interruptions are not considered. In other words, we assume that, if no out-of-bid situation takes place during an instance lifetime, its availability is 100%.

A. Bidding strategies: estimating cost and jobs' urgency

The first mechanism comprises bidding strategies and the calculation of the value of U. These are based on estimated price variations and job runtimes. More specifically, this mechanism aims to aid the process in two ways: (1) allow the broker to make informed decisions on how much to bid, a choice that directly influences the risk of failure and monetary spending; and (2) combine price information and a job's urgency factor, to decide the best point in time to start a new machine for a job, thus seeking to cover the period that will yield the minimum cost. The rationale behind combining these two pieces of information is to avoid hasty decision that may increase costs, i.e. to avoid commissioning new resources too early, at times when non urgent jobs can be postponed, or too late, when jobs will most likely miss their deadlines.

In our previous work [2], we have compared several runtime estimation policies and their impact on cost, deadline violations, and system utilization. A simple mechanism that computes the average runtime of two preceding jobs of the same user has performed consistently well. Therefore, in this work, we exclusively employ that technique.

We have evaluated 5 bidding strategies, which are listed on table I. Two of the strategies use historical information to compute the bid. In all cases, a window of one week worth of price history, individual to each instance type/OS/datacenter combination, is fed to the bidding strategy. The output of each strategy is the maximum price, in US dollars per hour, to be paid for one particular instance. The minimum bid granularity G is 0.001.

In all cases that can yield values lower than the current price, the broker uses the value of U to override the bid value, if necessary. Specifically, it applies the steps of Algorithm 1.

B. Hourly Checkpointing

Checkpointing consists of saving the state of a VM, application, or process, during execution and restoring the saved state after a failure to reduce the amount of lost work [13]. In

```
b ← compute bid;
U ← compute urgency factor;
P ← query provider for current price;
if U = 0 then
    if b <= P then
        b = P + G;
else
    schedule a bid check at T + U;
```

Algorithm 1: Bid check algorithm, which overrides the bid value or schedules a new check in the future

the context of virtual machines, the action of encapsulating execution and user customization state is a commonplace feature in most virtual machine monitors (VMM) [14]. Saving a VM state consists of serializing its entire memory contents to a persistent storage, thus including all applications and process running [15]. In our work, we assume that checkpointing a running application is the same as saving the state of an entire VM. The advantage of relying on VMM-supported checkpointing is that applications do not need to be modified to enable checkpointing-based fault-tolerance. However, it is necessary that cloud computing providers explicitly support such operation.

The technique considered in this work is a hourly-based VM checkpointing, where states are saved at hour-boundaries. This technique has been previously identified by Yi et al. [10] as the simplest and most intuitive, yet effective, form of dealing with the cost/reliability trade-off when running applications on spot instances. More specifically, taking a checkpoint on an hourly basis guarantees that only useful computational time is paid, given that spot instances are billed at an hour granularity and partial hours, in the case of failures, are not charged.

In this method, it is assumed that a checkpointed VM will only resume when the original spot request, which has a fixed bid and machine type, is in-bid again. No attempt is made to provision a new VM by submitting higher bids for the same machine type, or to bid for other types. This contrasts with our next solution, which considers relocating the saved state to a new space in order to hasten job completion.

C. Migration of persistent VM state

We propose a migration-based fault-tolerance mechanism on which the state of a VM is frequently saved on a global filesystem and upon an out-of-bid situation the state is relocated. The migration technique is very similar to checkpointing, as it comprises of taking a snapshot of the VM and using it to restore the computation upon a failure. But instead of waiting for the original request to be in-bid again, the algorithm aims to lease a new instance under new terms, and then restore the saved VM state into the new instance.

To decide new lease terms, i.e. where to allocate the remaining work of a saved job, the following decision-making process takes place: (1) the algorithm estimates the cost of leasing an instance of the same type for a higher price in the same datacenter; (2) then it considers leasing an instance of a different type on the same datacenter; (3) it also considers relocating the workload to another datacenter where a suitable VM may be leased for a cheaper price; finally, the chosen location is wherever is estimated to be cheaper to accomplish the remaining duration of the job. The overhead of restoring a failed VM in a distinct datacenter is assumed to be higher than when the same datacenter is chosen. This overhead is taken into account when making a relocation decision.

All computation in the VM is paused while the snapshot is being taken. The overhead of saving an instance state (the same as taking a checkpoint) is defined as the time to serialize a VM's memory snapshot into a file in a global filesystem. This value is different for each instance type, according to their maximum memory size. The exact values are computed as in the work of Sotomayor et al. [16], which provides a comprehensive model to predict the time to suspend and resume VMs. The times to suspend (i.e. save the state) and to resume (i.e. restore from the latest saved state) a spot instance with m MB of memory, are defined as per equations 2 and 3 respectively [16].

$$t_s = m/s \tag{2}$$

$$t_r = m/r \tag{3}$$

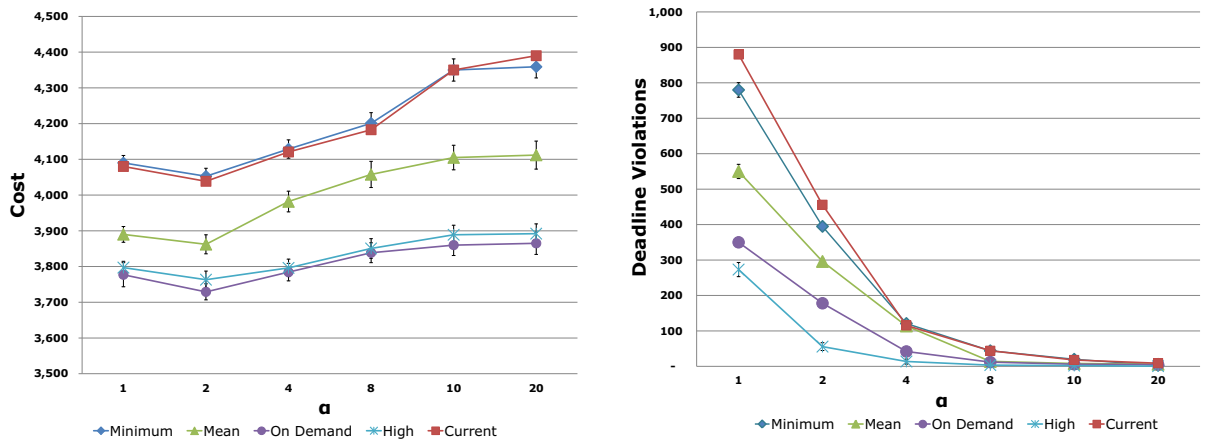
Values for s and r (rates, in MB/s, to write/read m MB of memory to/from a global filesystem) are also taken from [16], who obtained them from numerous experiments on a realistic testbed. Therefore, s is 63.67 MB/s, and r is 81.27 MB/s (to restore a state in the same datacenter). We assume half the rate (40.64 MB/s) when moving/restoring a VM state into/from a distinct datacenter.

D. Duplication of long jobs

We also propose a fault-tolerance mechanism that does not require any application- or provider-assisted technique, as it is the case of VM-based checkpointing and migration. With task duplication, we aim to evaluate a simpler method for rapid deployment of applications on spot instances using currently available cloud computing feature.

Similar to replication and migration, duplication of work aims to increase the chance of success in meeting deadlines when running longer jobs (greater than one hour) over a period of frequent price changes. Therefore, a duplication-based technique was implemented and evaluated.

This technique also relies on estimates of jobs runtimes. It creates one replica of each job that is expected to run for more than 1 hour. The replica is submitted to the same scheduling policy as the original job. The algorithm applies the same rules as it does to a regular job, but avoids choosing the the datacenter/type combination where the original job will run. Choosing a different combination for a replica is an obvious choice, since two jobs running on the same datacenter, using the same instance type, will certainly fail at the same time when the price increases.



(a) Monetary cost

(b) Deadline violations

Fig. 2. Effect of aggressive and conservative urgency estimation modifier (α) under various bidding strategies, without any fault-tolerance mechanism in action

TABLE II
FACTORS AND THEIR LEVELS

Factor	Possible values
Bidding Strategy	Minimum, Mean, On-demand, High, Current
α	1, 2, 4, 8, 10, 20
Fault tolerance mechanisms	None, Migration, Checkpointing, Job duplication

V. PERFORMANCE EVALUATION

In this section, we evaluate the proposed fault-aware resource allocation policy and the effect of its mechanisms, using trace-driven discrete event simulations. We quantify the performance of our policy based on three metrics, two absolute (monetary cost and deadline violations) and one relative (dollar per useful computation). We especially observe the interaction between these metrics, given that there is a known trade-off between them, i.e. assuring less violations usually means provisioning more resources, hence higher costs.

A. Experimental design

We have designed our experiments to study the influence of the following factors and their levels: (1) bidding strategy; (2) the value of the urgency factor modifier α; and (3) choice of fault-tolerance mechanism. The factors and their levels are listed on Table II.

Not all combinations of factors have been simulated; for example, there was little sense in combining the High bidding strategy with a fault-tolerance mechanism, given that the bidding fashion itself completely avoid failures. In total, 5952 experiments were executed. All values presented correspond to an average of 31 simulation runs. When available, error bars correspond to a 95% confidence interval. The simulator was implemented using the CloudSim framework [17].

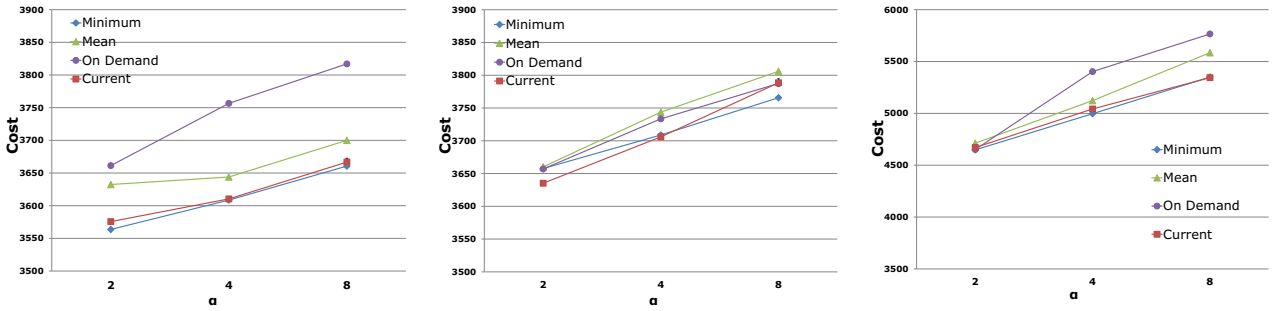
Cloud characteristics: We modeled the cloud provider after the features of Amazon EC2's US-EAST geographic region, which contains 4 datacenters. Instance types were modeled directly after the characteristics of available standard and high-CPU types The types available to be used are M1.SMALL (1 ECU), M1.LARGE (4 ECUs), M1.XLARGE (8 ECUs), C1.MEDIUM (5 ECUs), C1.XLARGE (20 ECUs). One ECU (EC2 Compute Unit) is defined as equivalent to the power of a 1.0-1.2 GHz 2007 AMD Opteron or 2007 Intel Xeon processor. A period of 100 days worth of pricing history traces has been collected comprising dates between July 5th, 2011 and October, 15th, 2011. These dates correspond to the available traces since Amazon EC2 has started offering distinct prices per individual datacenter, rather than per geographic region.

Workload: The chosen job stream was obtained from the LHC Grid at CERN [18], and is composed of grid-like embarrassingly parallel tasks. A total of 100,000 jobs are submitted over a period of seven days of simulation time, starting from a randomly generated time within the available price history. This workload is suitable to our experiments due to its bursty nature and for being composed of highly variable job lengths. These features require a highly dynamic computation platform that must serve variable loads while maintaining cost efficiency. The moldability parameters A and σ of each job are assumed to be known by the broker.

Originally, this workload trace did not contain information about user-supplied job runtime estimates and deadlines. User runtime estimates were generated according to the model of Tsafrir et al. [19]. A job's maximum allowed runtime corresponds to the runtime estimate multiplied by a random multiplier, uniformly generated between 1.5 (urgent jobs) and 4 (non-urgent jobs). A job's deadline corresponds to its submission time plus its maximum allowed runtime.

B. Effects of bidding strategies and urgency factor

In order to understand how our bidding strategies work, without any fault-tolerance mechanism in action, we have evaluated their effectiveness in a scenario where a failed



(a) Migration (b) Checkpointing (c) Job duplication

Fig. 3. Performance of migration, checkpointing and job duplication on monetary cost

job must be restarted from the beginning after a failure. In this experiment, we aimed at quantifying each strategy's performance when paired with different values of α.

Figure 2 shows the effect of most aggressive ($\alpha = 1$) to most conservative ($\alpha = 20$) urgency estimations under various bidding strategies. In these circumstances, bidding strategies that produce higher bids tend to perform better, both in terms of cost and deadline violations. In particular, we have observed that the On-demand strategy avoids failures due to minor price increases, as well as avoids incurring the cost of high prices above the on-demand price. This fact can be noticed in the performance comparison between On-demand and High, which incurs extra cost due its very high bid. As expected, given that there was no fault-tolerance, strategies that aim at bidding low values experience the most failures, hence more work had to be redone from start, which consequently led to higher costs.

The value of α significantly influences both cost and deadline violations, consistently over all bidding strategies. Figure 2(a), indicates an optimal value of 2, which yields the lower costs, although for most bidding strategies, the difference between 1 and 2 is not statistically significant. Regarding the deadline metric, 1 and 2 lead to many more deadline violations. This is due to the fact that lower values of α cause the algorithm to postpone more decisions, which in turn often leads to the inability of provisioning resources "at the last minute". Conservative values, on the other hand, lead to virtually no violations, but higher costs.

C. Migration, checkpointing, and job duplication

Our results also demonstrate the positive effects of the studied fault-tolerance mechanisms when paired with bidding strategies and urgency factor estimation. Figure 3 shows a comparison of migration, checkpoint and job duplication on the cost metric. We only show values of α of 2, 4 and 8, which yield the best costs in all cases. An interesting fact is that migration performs better when paired with bidding strategies that choose lower bid values, such as Minimum and Current, while checkpointing benefits from higher bid values, such as Mean and On-demand. This behaviour is coherent with the features of each mechanism. Migration tends to have more choices after an out-of-bid situation given its ability to choose other types of instances from multiple datacenters. Checkpointing, on the other hand, is bound to a persistent request, and will benefit from a higher chance of being in-bid most of the time.

Job duplication was found to perform poorly in all cases, yielding higher costs when compared to the case when no fault-tolerance exists. It's merit however, lies on its simplicity and the capability of replicating jobs across multiple datacenters. Therefore, it can be useful in cases where an extra level of redundancy is required.

Figure 4 presents a summary of best combinations of strategies discovered in our simulations. Overall, the migration technique, along with the Minimum bidding strategy and $\alpha = 2$ produced the lower cost. However, $\alpha = 8$ produced the least number of deadline violations (30 out of 100,000 jobs). These results confirm that the trade-off between cost and deadline violations applies in this case.

In summary, our results demonstrate that the interaction of factors can influence the exact choice of bidding strategy, α, and fault-tolerance mechanism. It is expected that, in absolute

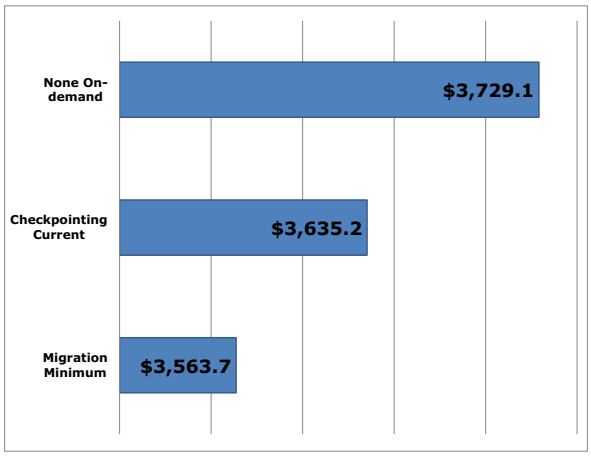


Fig. 4. Most economical combinations of bidding strategy and fault-tolerance mechanism

TABLE III
ANALYSIS OF THE DOLLARS PER USEFUL COMPUTATION METRIC

Rank	Fault tolerance	Bidding strategy	α	Dollars per useful computation	Worsening related to best (%)
1	Migration	Minimum	2	0.03578	0
2	Migration	Current	2	0.03588	0.29
3	Migration	Minimum	4	0.03613	0.978
4	Migration	Current	4	0.03614	1.006
5	Migration	Mean	2	0.03641	1.736
6	Checkpointing	Current	2	0.03647	1.881
7	Migration	Mean	4	0.03648	1.932
8	Migration	Minimum	8	0.03661	2.279
9	Checkpointing	On-demand	2	0.03663	2.330
10	Checkpointing	Mean	2	0.03666	2.412
...					
18	None	On-demand	2	0.03736	4.224

terms, more conservative urgency factors will lead to less deadline violations and a greater cost. To help gauge a more precise metric, we define dollars per useful computation as the ratio between the total cost and the number of jobs that finished within their deadlines. Table III ranks the 10 best factor combinations according this metric. The combinations that employ migration rank consistently superior, which makes these combinations good candidates for environments where a strict meeting of deadlines is expected.

VI. Conclusions and Future Work

In this work, he have proposed a multifaceted resource provisioning policy that reliably manages a pool of intermittent spot instances. Our policy contains multiple mechanisms, including 5 bidding strategies, an adjustable urgency factor estimator, and 3 fault-tolerance approaches.

We have performed extensive simulations under realistic conditions that reflect the behaviour of Amazon EC2, via a history of its prices. Our results demonstrate that both costs savings and stricter adherence to deadlines can be achieved when properly combining and tuning the policy mechanisms. Especially, the fault-tolerance mechanism that employs migration of VM state providers superior results in virtually all metrics.

Currently, the cloud computing spot market is still in its infancy. Therefore, many challenges have not been encountered, given the short history and relatively low variability of Amazon EC2 prices. In this sense, we plan to further improve our policy by devising bidding strategies that will perform well in environments with highly variable price levels and more frequent changes. We expect fault-tolerance to be even more crucial in such scenarios. We also plan to lean towards provider-centric research, by studying the challenges involved in setting spot prices under various demand patterns.

Acknowledgement

This work is partially supported by grants from the Australian Research Council (ARC), and the Australian Department of Innovation, Industry, Science and Research (DIISR). We also would like to thank ej-technologies GmbH for providing a license of jProfiler for use in the CloudSim project. Special thanks to Yoganathan Sivaram, Adel Toosi, Deepak Poola, Bahman Javadi, and the anonymous reviewers, who helped improve the quality of this paper.

References

[1] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. USENIX Association, 2010, pp. 7–7.

[2] W. Voorsluys, S. Garg, and R. Buyya, "Provisioning spot market cloud resources to create cost-effective virtual clusters," in *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2011.

[3] Amazon Web Services LLC, "Amazon Web Services," http://aws.amazon.com.

[4] J. Varia, "Best practices in architecting cloud applications in the aws cloud," in *Cloud Computing, Principles and Paradigms*, R. Buyya, J. Broberg, and A. M. Goscinski, Eds. Wiley, 2011, ch. 18, pp. 457–490.

[5] Q. Zhang, E. Gürses, R. Boutaba, and J. Xiao, "Dynamic resource allocation for spot markets in clouds," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*. USENIX Association, 2011.

[6] A. Andrzejak, D. Kondo, and S. Yi, "Decision Model for Cloud Computing under SLA Constraints," INRIA, Tech. Rep., 2010. [Online]. Available: http://hal.archives-ouvertes.fr/inria-00474849/

[7] M. Mazzucco and M. Dumas, "Achieving performance and availability guarantees with spot instances," in *Proceedings of the 13th International Conferences on High Performance Computing and Communications (HPCC)*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2011.

[8] L. Ausubel and P. Milgrom, "The lovely but lonely vickrey auction," in *Combinatorial Auctions*, P. Cramton, Y. Shoham, and R. Steinberg, Eds. MIT Press, 2006, ch. 1, pp. 17–40.

[9] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "Deconstructing amazon ec2 spot instance pricing," Technion Israel Institute of Technology, Tech. Rep. CS-2011-09, 2011.

[10] S. Yi, D. Kondo, and A. Andrzejak, "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud," in *2010 IEEE 3rd International Conference on Cloud Computing*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2010, pp. 236–243.

[11] A. B. Downey, "A model for speedup of parallel programs," Berkeley, CA, USA, Tech. Rep., 1997.

[12] W. Cirne and F. Berman, "A Model for Moldable Supercomputer Jobs," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2001.

[13] C. Kintala, "Checkpointing and its applications," in *Proceedings of the 25th International Symposium on Fault-Tolerant Computing*. Los Alamitos, CA, USA: IEEE Comput. Soc., 1995, pp. 22–31.

[14] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Hypervisor-assisted application checkpointing in virtualized environments," in *Proceeding of the 41st IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, june 2011, pp. 371 –382.

[15] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume," in *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2002, pp. 40–46.

[16] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Resource Leasing and the Art of Suspending Virtual Machines," in *Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications (HPCC)*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2009, pp. 59–68.

[17] R. Buyya, R. Ranjan, and R. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proceeding of the 7th International Conference on High Performance Computing & Simulation (HPCS)*. Los Alamitos, CA, USA: IEEE Comput. Soc., 2009, pp. 1–11.

[18] D. Feitelson, "Parallel workloads archive," http://www.cs.huji.ac.il/labs/parallel/workload.

[19] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Modeling User Runtime Estimates," in *In Processing of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. Springer-Verlag, 2005, pp. 1–35.