Application-Centric Resource Provisioning for Amazon EC2 Spot Instances

Sunirmal Khatua[1] and Nandini Mukherjee[2]

[1] University of Calcutta, Kolkata, India
skhatuacomp@caluniv.ac.in
[2] Jadavpur University, Kolkata, India
nmukherjee@jdvu.ac.in

Abstract. In late 2009, Amazon introduced spot instances to offer their unused resources at lower cost with reduced reliability. Amazon's spot instances allow customers to bid on unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current spot price. The spot price changes periodically based on supply and demand of spot instances, and customers whose bid exceeds it gain access to the available spot instances. Customers may expect their services at lower cost with spot instances compared to on-demand or reserved. However the reliability is compromised since the instances (IaaS) providing the service (SaaS) may become unavailable at any time without any notice to the customer. In this paper, we study various checkpointing schemes to increase the reliability over spot instances. Also we devise a novel checkpointing scheme on top of application-centric resource provisioning framework that increases the reliability while reducing the cost significantly.

Keywords: resource provisioning, spot instances, checkpointing.

1 Introduction

The era of cloud computing provides high utilization and high flexibility of managing the computing resources. The elasticity and on demand availability features of cloud computing ensure high utilization of resources. Furthermore, resources can be availed from templates that enforce standards so that resources can be used with best management considerations without prior knowledge. Therefore, flexibility of managing the computing resources is also high in a cloud environment. The cloud computing service models incorporate Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides raw computing resources with different capacity in the form of Virtual Machines (VM). Cloud Service Providers (CSP), like Google [16], Amazon [15] etc. provide these services and charge prices against these services from the Cloud Service Users (CSU). Among many such providers, Amazon defines the capacity of resources in the form of different instance types [11] based on storage, compute unit and I/O performance. The cost of these instance types depends on the purchasing models [12] defined by Amazon namely on-demand, reserved and spot.

$On-demand$ instances let one pay for compute capacity by the hour with no long-term commitments or upfront payments. However, with on-demand instances one may

F. Wolf, B. Mohr, and D. an Mey (Eds.): Euro-Par 2013, LNCS 8097, pp. 267–278, 2013.

not have access to the resources immediately due to high demand for a specific instance type in a specific availability zone. On the other hand, *reserved instances* facilitate the client to make a low, one-time, upfront payment for an instance, reserve it and get significant discount on hourly charge over on-demand instances. Reserved instances are always available for the duration for which the clients reserve. In contrast with the above two policies, where rates are fixed, *spot instances* provide the ability for customers to purchase compute capacity with no upfront commitment and at a variable hourly rate with a customer-defined upper bound (bid) on the rate. Spot instances are available only during the time when the spot price is below the customer defined bid.

Thus spot instances make the resources unreliable in nature and inappropriate for long running jobs like image processing, gene sequence analysis etc. At the same time, they offer the opportunity to accomplish such jobs at a much lower cost than on demand or reserved policies. Clearly, checkpointing (saving partially completed jobs to be resumed latter) may be a good option to make a tradeoff between the cost and reliability. Again, the time of taking a checkpoint and the frequency of taking the checkpoints directly affect the cost and reliability. Sufficient research effort is needed to properly set the time and frequency of taking the checkpoints.

The rest of the paper is organized as follows. A brief review of the related works is presented in Section 2. An overview of the application centric resource provisioning framework is given in Section 3. Section 4 deals with the existing checkpointing schemes for spot instances while a proposed checkpointing scheme for the application centric resource provisioning framework is described in Section 5. A simulated result for comparing the proposed checkpointing scheme with existing ones is presented in Section 6. Finally, we conclude with a direction of future work in Sections 7.

2 Related Work

During the last couple of years, a lot of works [1] [8]-[9] concentrate on the cloud management aspect from the economic point of view. Most of them adapt a middleware based (broker) approach to optimize the resource requirement for a given cloud application. In our previous work [1], we provide a novel framework for such a middleware. It identifies the key components of the middleware for auto deploying, auto scaling, providing robustness and availability of heterogeneous cloud applications. A model for optimal cloud resource scheduling based on stochastic integer programming technique is proposed in [8]. A similar technique is also used in [9] to optimize the resource requirement of a cloud application. This work tries to minimize the total provisioning cost by adjusting the tradeoff between the reserved and on-demand resource provisioning plans.

Some research works [2]-[6] also consider Amazon EC2 spot instances [13] for providing economic benefit to cloud service users considering availability and reliability. Various checkpointing techniques have been discussed in [2] to provide reliability with Amazon spot instances at lower cost. In this paper, we study some of these techniques and evaluate their performances. We also investigate the effectiveness of application centric resource provisioning framework [1] for actively monitoring the deployed spot instances for an application and for taking necessary actions as the spot intances become unavailable or the spot price changes. Finally, we propose and evaluate a novel

checkpointing scheme for the application centric resource provisioning framework that outperforms all the checkpointing schemes defined in [2].

3 Application-Centric Resource Provisioning Framework

An Application-centric resource provisioning framework along with the unified definition of an application is proposed in [1]. A brief description of functioning of the application centric resource provisioning framework is depicted in Figure 1. The framework consists of two key subsystems namely Provisioning subsystem and Monitoring subsystem.

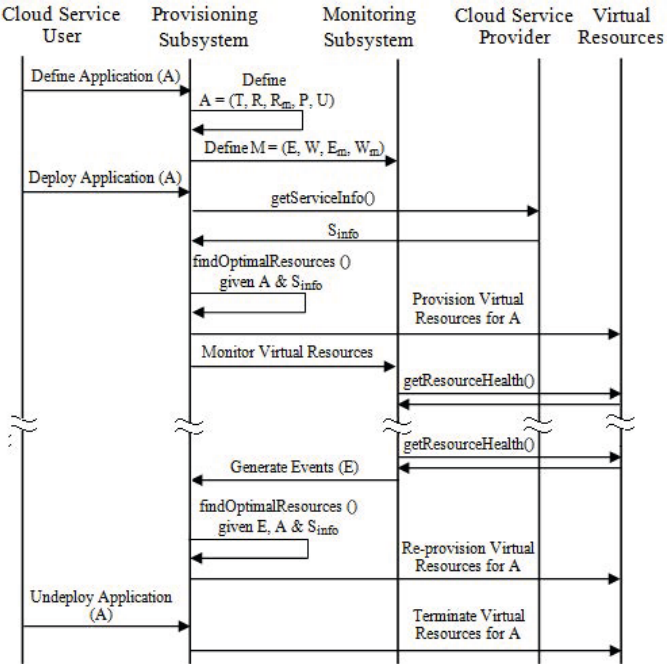

Fig. 1. Resource provisioning algorithm

3.1 Provisioning Subsystem

The provisioning subsystem determines optimal provisioning of virtual resources for an application A) satisfying the policies (P) specified for it. The application's required service level is stored in the policy (P). The provisioning subsystem queries various providers to get information about their offered services (S_{info}). S_{info} consists of provider id, service id, QoS id and the associated cost. The provisioning subsystem uses P (desired service level), S_{info} and an optimization algorithm to find the optimal resource requirement for the application while maintaining the desired service level.

3.2 Monitoring Subsystem

The Monitoring subsystem implements a feedback system to inform the provisioning subsystem about the current state of the deployed application. The monitoring subsystem actively monitors the state of the deployed application and generates various events [1] to designate a change in the application state. Once an event is generated, the monitoring subsystem sends the event to the provisioning subsystem. Once an event(E) is received, the provisioning subsystem analyzes the event and uses E, P, S_{info} and an optimization algorithm for reprovisioning the application onto appropriate resources.

4 Checkpointing Schemes for Amazon EC2 Spot Instances

In this paper multiple providers of application centric resource provisioning are not considered. Instead, we consider the spot market of Amazon EC2 public CSP only. The concept can be generalised to any CSP supporting spot model.

As discussed earlier, the variable price of spot instances makes them an important consideration for optimizing resource requirement for an application. However, their volatile nature makes them inherently unreliable and hence the optimization algorithms become more challenging than the other instances.

4.1 Characteristics of Spot Instances

Before dealing with the challenges of optimizing the use of spot instances, let us summarize the characteristics of Amazon EC2 spot instances [13] as listed below:

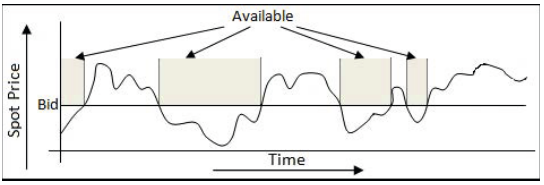

Fig. 2. Resource provisioning algorithm

- Spot instances are available when the user's bid exceeds the current spot price (refer Fig. 2).
- Spot instances are terminated (becomes unavailable) without any notification to the user whenever the current spot price exceeds the user's bid.
- The price per instance-hour for a spot instance is set at the beginning of each instance-hour. Any change to the spot price will not be reflected until the next instance-hour begins.
- Amazon will not charge the last partial hour if the spot instance is terminated due to out-of-bid situation. However Amazon will charge the full hour if the user terminate the instance forcefully.
- Amazon provides the history of spot prices of a spot instance at a specific availability zone for the last 3 months free of cost.

4.2 Existing Checkpointing Schemes for Spot Instances

The characteristics of spot instances make them appealing for long running jobs with divisible workloads [10]. Clearly, taking checkpoints at regular interval increases the utilization of spot instances. Various existing checkpointing schemes can be adopted for saving the completed tasks and resuming the remaining tasks as and when the spot instances become available. The checkpointing schemes proposed in [2] are briefly described below:

1. No Checkpointing (NONE): Checkpoints are not taken and all the partially completed tasks for a job are required to be repeated after every out-of-bid events.

2. Optimal Checkpointing (OPT): Checkpoints are taken just prior to the out-of-bid events. Clearly, it will save the maximum number of tasks out of each available interval for a given instance type and a user's bid.

3. Hourly Checkpointing (HOUR): Checkpoints are taken just prior to the beginning of next instance hour. Since Amazon is not charging any partial hour, this scheme will save as much tasks as the user is paying.

4. Rising edge-driven Checkpointing (EDGE): Checkpoints are taken after every increase (rising edge) of the current spot price.

5. Adaptive Checkpointing (ADAPT): Checkpoints are taken or skipped at regular intervals based on the expected recovery time for skipping (R_{skip}) or taking (R_{take}) a checkpoint. The estimation of R_{skip} and R_{take} is given in the equations 1 and 2. Here r is the task recovery time, t_p is the present time, $f(t)$ is the probability density function of out-of-bid events, t_r is the time needed to complete a job, t_c is the time needed to take a checkpoint and $T(t, t_p)$ is the expected execution time for a job of length t started at time t_p. Checkpoints are taken when R_{skip} is greater than R_{take}.

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

$$R_{take}(t, t_p) = \sum_{k=0}^{t_r-1} (k+r))\, f(k+t_p) + t_c \sum_{k=t_r}^{\infty} f(k+t_p) + T(t, t_p - t) \sum_{k=0}^{t_c-1} f(k+t_p) \tag{2}$$

$$T(t, t_p) = (t \sum_{k=t}^{\infty} f(k + t_p) + \sum_{k=0}^{t-1} (k + r)\, f(k + t_p))/(1 - \sum_{k=0}^{t-1} f(k + t_p)) \tag{3}$$

Out of the above five checkpointing schemes, NONE and OPT provide two extreme results without any practical value. They are used to provide comparative study of the other realistic checkpointing schemes.

5 A Novel Checkpointing Scheme over Application-Centric Resource Provisioning Framework

In this section, we propose a novel checkpointing scheme for spot instances on top of application-centric resource provisioning framework. For the purpose, we devise a new

event generation scheme that deals with spot instances. The new checkpointing scheme is targeted to achieve performance comparative to OPT checkpointing scheme described above. Before describing the scheme, we introduce a modified event generation scheme for our application-centric resource provisioning framework.

5.1 Event Generation Scheme for Spot Instances

The event generation schemes proposed in [1] is extended to include new events that support spot instances. As discussed in Section 4.1, the availability of spot instances depends on the current spot price and the user defined bid. Also, spot instances become unavailable without prior notification to the clients that makes them inherently unreliable. The reliability can be increased by taking checkpoints (saving completed tasks) during the available periods. However, the time and frequency of taking checkpoints affect the reliability as well as job completion time and cost.

Accordingly, in this paper we propose a new event generation scheme to handle spot instances. Three events are proposed, namely E_{ckpt}, $E_{terminate}$ and E_{launch}. E_{ckpt} is used for taking checkpoint, $E_{terminate}$ is used to terminate a spot instance forcefully and E_{launch} is used to relaunch a previously terminated spot instance. We define two bid values for the purpose - one for the application (A_{bid}) and other for the spot instance (S_{bid}). S_{bid} is sufficiently large and is used in the request for spot instance. Clearly, the value is maintained at such a high level, that Amazon will never terminate the spot instances due to out-of-bid situation. On the other hand, A_{bid} is used by the monitoring subsystem to maintain user's budget.

The monitoring subsystem actively monitors the current spot price and generates the two events, E_{ckpt} and $E_{terminate}$, for the provisioning subsystem. On the basis of these two events, the provisioning subsystem either takes a checkpoint or terminate the corresponding spot instance respectively. However, to increase the performance, the monitoring subsystem will query the current spot price only at specific points of time called decision points. Since the cost of spot instance is not changed during an instance hour and is fixed at the beginning of that instance hour, the decision points should be relative to the beginning of the next instance hour. Accordingly, we define two decision points just prior to each hour boundary as follows:

$$t_{cd} = t_h - t_c - t_w \tag{4}$$

$$t_{td} = t_h - t_w \tag{5}$$

where t_{cd} and t_{td} are the decision points for checkpointing and terminating a spot instance. t_h is an hour boundary, t_c is the time needed to take a checkpoint and t_w is the waiting time to get the current spot price. The monitoring subsystem will generate E_{ckpt} at t_{cd} if the current spot price exceeds A_{bid} and will generate $E_{terminate}$ at t_{td} if the current spot price is still above the A_{bid}. It will generate E_{launch} at the start of each available period of a spot instance with respect to A_{bid}.

5.2 The Application-Centric Checkpointing Scheme

In this section, we propose a checkpointing scheme on top of the application centric resource provisioning framework, called Application Centric Checkpointing(ACC). ACC is based on the event generation scheme discussed in the previous subsection and is described by the sequence diagram shown in Fig. 3.

The following unified definition can be used for an application with divisible workloads to be run on spot:

$$A = (T, R, R_m, P, U, M) \tag{6}$$

where $T = \{t_1\}$
$R = \{r_1, r_2\}$, r_1.provider = ec2, r_1.type = spot instance,
r_1.size = $< instance\_type >$
r_2.provider = ec2, r_2.type = EBS, r_2.size = 1GB
$R_m = \{ r_1 \rightarrow t_1, r_2 \rightarrow t_1 \}$
$P = \{ sla \}$

$$M = (E, W, E_m, W_m) \tag{7}$$

where $E = \{E_{ckpt}, E_{terminate}, E_{launch}\}$, $threshold\ for\ all\ events =< A_{bid} >$
$E_{launch}.bid =< S_{bid} >$
$W = \{W_{start}, W_{ckpt}, W_{terminate}, W_{launch}\}$
W_{start} = { Launch spot; Mount EBS; Copy job to EBS; Start job },
W_{ckpt} = {Save results to EBS},
$W_{terminate}$ = {Terminate spot} &
W_{launch} = { Launch spot; Mount EBS; Resume tasks },
$E_m = \{E_{ckpt} \rightarrow r_1,\ E_{terminate} \rightarrow r_1, E_{launch} \rightarrow r_1\})$
$W_m = \{W_{ckpt} \rightarrow E_{ckpt}, W_{terminate} \rightarrow E_{terminate}, W_{launch} \rightarrow E_{launch}\}$

The Elastic Block Storage (EBS) [14] is used to save the completed tasks during checkpoint. The parameters $instance\_type$, A_{bid} and S_{bid} can be set either manually by the end user or by some optimization or greedy algorithms.

The provisioning subsystem starts an application (job) by executing W_{start} workflow for that application. The W_{start} workflow launches a spot instance as per the specification of the resource r_1 and an EBS volume as per the specification of the resource r_2. The workflow then mounts the EBS volume to the spot instance, copy the job from the application repository to the EBS and starts the job.

Once the application is deployed, EC2 starts charging for the resources. The monitoring subsystem calculates t_{cd} and t_{td} as per Equ. 4 & 5 for the current hour boundary. At t_{cd} the monitoring subsystem retrieves the current spot price(P). If P exceeds A_{bid}, it generates E_{ckpt} event for the provisioning subsystem. On receiving E_{ckpt} event, the provisioning subsystem executes W_{ckpt} workflow. The W_{ckpt} workflow just saves the results (the completed tasks) to the EBS volume. The monitoring subsystem also retrieves the current spot price(P) at t_{td}. If P still exceeds A_{bid}, it generates $E_{terminate}$ event for the provisioning subsystem. On receiving $E_{terminate}$ event, the provisioning subsystem executes $W_{terminate}$ workflow. The $W_{terminate}$ workflow terminates the

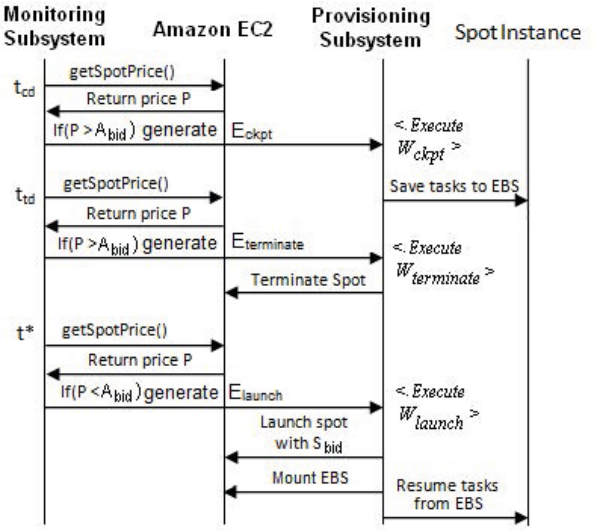

Fig. 3. Application Centric Checkpointing Scheme

spot instance forcefully. The monitoring subsystem repeats the above procedure till P does not exceed A_{bid} at t_{td} for all the subsequent hour boundaries.

If the instance is terminated at some t_{td}, the monitoring subsystem will have to query for the current spot price to determine the next available period at some specific instance of time(t*). However, the frequency of making the query is defined by the end user which may affect the job completion time slightly. At the start of the new available duration, the monitoring subsystem generates E_{launch} event for the provisioning subsystem. On receiving E_{launch} event, the provisioning subsystem executes W_{lauch} workflow. The W_{launch} workflow launches a new spot instance as specified in r_1, mount the existing EBS volume to that instance and resume the remaining tasks of the job.

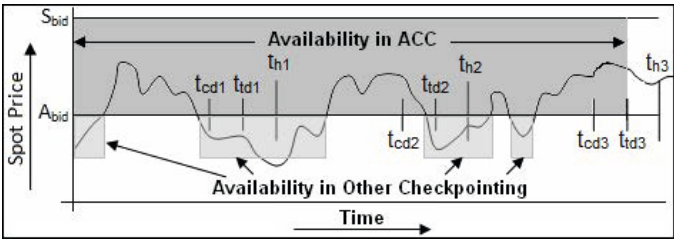

Fig. 4. Decision Points for Event Generation

The novelty of the scheme is illustrated in Fig. 4. ACC will generate neither E_{ckpt} nor $E_{terminate}$ for the hour boundary t_{h1} since the current spot price is bellow A_{bid} at both the decision points. That means, it will neither take a checkpoint nor terminate the

spot instance at t_{h1}. It will generate E_{ckpt} but not $E_{terminate}$ for the hour boundary t_{h2} since the current spot price is above A_{bid} at t_{cd2} and below A_{bid} at t_{td2}. That means, it will take a checkpoint but will not terminate the spot instance at t_{h2}. Similarly, for the hour boundary t_{h3}, it will generate both E_{ckpt} and $E_{terminate}$ since the user will have to pay above A_{bid} for the next hour. So, it will take a checkpoint as well as terminate the spot instance at t_{h3}. Clearly availability is increased and more continuous in ACC compared to other checkointing schemes as shown in Fig. 4.

6 Implementation and Evaluation

In this section we analyze and compare our proposed ACC checkpointing scheme with the existing checkpointing schemes. The experiments have been carried out on 64 spot instance types using the same data set, parameters, algorithms and assumptions used in the simulator [26].

We obtain the simulation result for $job\ completion\ time$, $total\ monetary\ cost$ and the $product\ of\ monetary\ cost$ x $completion\ time$ for all the EC2 instance types. To simplify the discussion, we present the result of a linux based extra large (m1.xlarge) instance type in the eu-west-1 region. We concentrate on the performance of our proposed ACC checkpointing scheme compared to the theoritical optimal checkpointing scheme, OPT. We also include NONE, HOUR, EDGE and ADAPT checkpointing schemes in our result for completeness.

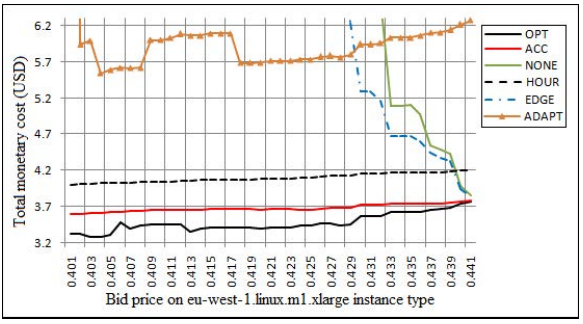

Fig. 5. Total monetary cost of Job completion

Fig 5 shows the comparison of total monetary cost needed to complete a job of length 500 minutes under different user's bid(A_{bid}) from \$0.401 to \$0.441. The result shows that ACC reduces the job completion cost significantly over the other realistic checkpointing schemes. However the cost is increased by 5.94% on average (min 0.33%, max 10.30%) compared to OPT scheme. This is because the OPT scheme guarantees payment of the actual progress of the job as well as executing some fraction of the job free of cost for the partial hours.

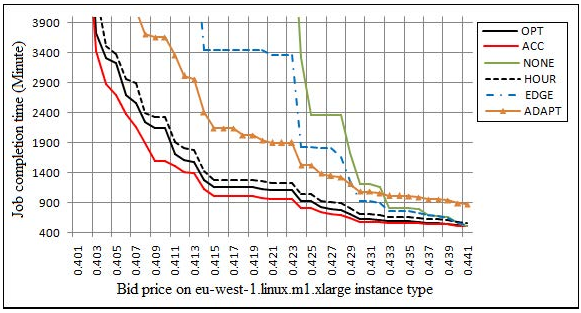

Fig. 6. Job completion time

In Fig. 6 we illustrate the comparison of various checkpointing schemes for the metric $job\ completion\ time$. Here we observe that ACC scheme outperforms all the checkpointing schemes including OPT. This is because ACC allows the job to continue even when the current spot price exceeds A_{bid} in between a t_{td} and the previous hour boundary (refer to Fig. 4). With OPT, the available duration is fragmented as shown in Fig. 2 while ACC allows the spot instance to be continuously available till t_{td3} a shown in Fig. 4 without affecting the job completion cost. That means the interruption to job execution is much less in ACC compared to OPT. In fact the ACC scheme reduces the job completion time by an average value of 10.77% over the OPT scheme.

We plot the comparative study for the $product\ of\ monetary\ cost$ x $completion\ time$ in Fig. 7. Here also we observe that the ACC scheme reduce this metric by an average value of 5.56% over the OPT scheme.

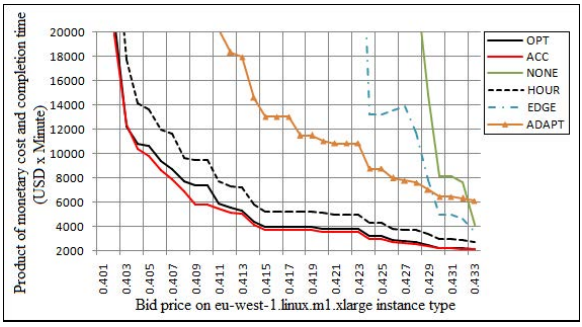

Fig. 7. Product of total cost and completion time

To gain confidence in our result, we have computed the average values of the above mentioned metrics for different bid values on all the 64 instance types. A sample of 15 difference instance types for the metric $product\ of\ monetary\ cost$ x $completion\ time$ is shown in Fig. 8. For these 15 instance types, a gain of 4.03% for ACC over OPT is observed. We also observe that such percentage gain is increased for costly instance types.

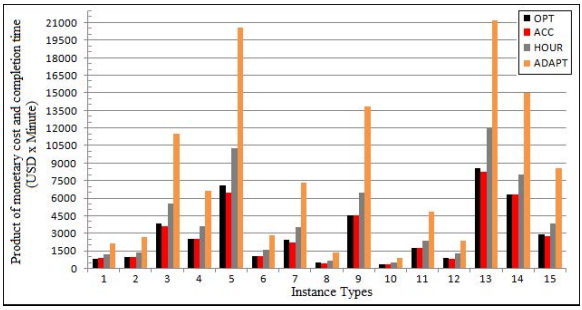

Fig. 8. Product of cost and completion time for different instance types

In the previous research work [2], the authors conclude that OPT is the optimal checkpointing scheme and none of the practical schemes can perform better than OPT. That is true only if we use the same bid values for launching the spot instance and executing the checkpoint. However, our proposed ACC checkpointing scheme perform very close to OPT or even better than OPT (for time and product metrics) by separating these two bid values. Thus ACC outperforms all the existing checkpointing schemes for spot instances. ACC achieves such performance gain by increasing availability at the same cost as shown in Fig. 4.

7 Conclusion and Future Work

Checkpointing plays an important role in reliability of job execution over EC2 spot instances. In this paper, we propose a checkpointing scheme on top of application-centric resource provisioning framework that not only increases the reliability but also reduces the cost significantly over the existing checkpointing schemes. The job completion cost under the proposed scheme is very close to the optimal checkpointing scheme. It performs better than all the practical checkpointing schemes for spot instances. In future, we want to investigate more on finding the optimal bid (A_{bid}) and the corresponding instance type for a given job.

References

1. Khatua, S., Ghosh, A., Mukherjee, N.: Application-centric Cloud Managemengt. In: 9th IEEE/ACS International Conference on Computer Systems and Applications(AICCSA), pp. 9–15 (2011)
2. Yi, S., Andrzejak, A., Kondo, D.: Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances. IEEE Transactions on Services Computing 5, 512–524 (2011)
3. Voorsluys, W., Buyya, R.: Reliable Provisioning of Spot Instances for Compute-intensive Applications. In: 26th IEEE AINA, pp. 542–549 (2012)
4. Javadi, B., Thulasiramy, R.K., Buyya, R.: Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In: 4th IEEE UCC, pp. 219–228 (2011)
5. Yi, S., Zafer, M., Kang-Won, L.: Optimal bidding in spot instance market. IEEE INFOCOM, 190–198 (2012)

6. Mazzucco, M., Dumas, M.: Achieving Performance and Availability Guarantees with Spot Instances. In: 13th IEEE HPCC, pp. 296–303 (2011)
7. Padala, P., et al.: Adaptive control of virtualized resources in utility computing enironments. In: Proceedings of EuroSys (2007)
8. Li, Q., Guo, Y.: Optimization of Resource Scheduling in Cloud Computing. In: 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 315–320 (2010)
9. Chaisiri, S., Lee, B., Niyato, D.: Optimization of Resource Provisioning Cost in Cloud Computing. IEEE Transactions on Services Computing (2011)
10. Yang, Y., Casanova, H.: Umr: A multi-round algorithm for scheduling divisible workloads. IPDPS 24 (2003)
11. Amazon EC2 Instance Types, http://aws.amazon.com/ec2/instance-types/
12. Amazon EC2 Purchasing Options, http://aws.amazon.com/ec2/purchasing-options/
13. Amazon EC2 spot instances, http://aws.amazon.com/ec2/spot-instances/
14. Elastic Block Storage, http://aws.amazon.com/ec2/ebs/
15. Garfinkel, S.: An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Tech. Rep. TR-08-07, Harvard University (2007)
16. Google Cloud Offering, http://cloud.google.com/products/
17. Barham, P., et al.: Xen and the Art of Virtualization. In: Proceedings of the 19th ACM symposium on Operating Systems Principles (2003)
18. Wolsky, R., et al.: Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems. Tech. Rep. 2008-10, University of California, Santa Barbara (2008)
19. Zabbix: an enterprise-class open source distributed monitoring solution for networks and applications, http://www.zabbix.com/
20. Harmer, T., et al.: An application-centric model for cloud management. In: Proceedings of 6th World Congress on Services, pp. 439–446 (2010)
21. Lim, H.C., et al.: Automated control in cloud computing: challenges and opportunities. In: Proceedings of the 1st workshop on Automated control for datacenters and clouds, Spain (2009)
22. Mills, T.C.: Time Series Techniques for Economists. Cambridge University Press (1990)
23. Buyya, R., et al.: Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: 10th IEEE International Conference on High Performance Computing and Communications, pp. 5–13 (2008)
24. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation of Computer Systems 27, 871–879 (2011)
25. Shao, J., Wang, Q.: A Performance Guarantee Approach for Cloud Applications Based on Monitoring. In: 35th IEEE Annual Computer Software and Applications Conference Workshops, pp. 25–30 (2011)
26. Checkpointing Simulator for spot instances, http://spotckpt.sourceforge.net