The Journal of Supercomputing (2018) 74:5980–6032
https://doi.org/10.1007/s11227-018-2509-0

A survey on optimal utilization of preemptible VM instances in cloud computing

Ashish Kumar Mishra[1] · **Brajesh Kumar Umrao**[1] · **Dharmendra K. Yadav**[1]

Published online: 30 July 2018

Abstract

In this article, an extensive survey on optimal utilization of preemptible instances in cloud is presented. Different techniques used in state-of-the-art research for efficient utilization of spot instances have been classified and categorized in the paper. To the best of our knowledge, this is the first attempt of its kind. With the continuing growths in cloud computing, researchers and business personnels are exploiting the services provided by the cloud computing to reduce their operational cost. Users can share resources in the cloud with the help of virtualization. Virtualization provides abstraction of cloud to the users by hiding the complexity of inherent software and hardware present in the cloud. It increases the likelihood of running multiple operating systems (OSs) on a single physical machine with sharing of hardware resources. Each OS can be considered as a virtual machine (VM) installed on a physical machine. Based on subscription model, VMs can be classified into three types: reserved VMs, on-demand VMs and spot VMs. Spot instances are also known as preemptible VM instances. Spot instances are used as reduced cost resources at the risk of reliability. To utilize spot instances, users have to bid for them. Users will able to get the spot instances only if the biding price is greater than the spot instance price. As soon as the bid price becomes less than the spot price, the cloud provider will revoke the VMs (SIs). This survey aims to find the ways, one can efficiently utilize spot instances for executing the tasks with optimized cost and time.

Keywords Cloud computing · Virtual machines · Spot instance · Bid price · Spot price · Fault tolerance · Checkpointing

✉ Ashish Kumar Mishra
ashish.rcs51@gmail.com; 2015rcs51@mnnit.ac.in

[1] Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Allahabad 211004, India

1 Introduction

We live in the era of web that facilitates people to be in contact with their colleagues from their own place. Immense number of users are hitting a website for accessing Internet or web. This tremendous load will need extra support of resources in terms of both hardware and software for continuous service. The increased load will not remain permanently. So, it is inefficient to buy resources; rather, one can acquire resources on rent basis to pay only for the consumed quantity and time. In this situation, cloud computing plays a significant role. Pool of resources are available in the cloud and a user can get that by paying only for the utilized magnitude and duration [1].

There are many cloud providers to offer resources for users. Resources available in the cloud can be requested either on reserved or on-demand basis. If any resource remains free, i.e., not demanded through reserved or on-demand request, such free resources can be pooled and send to the market (spot market) for bidding. Reserved instances are useful for planned request, while on-demand request can be done for sudden request. End time of the job to be submitted is needed for on-demand requests. In case of unavailability of the end time of task, bidding for spot instances in the spot market can be an effective option. The main advantage of using preemptible VM instances[1] is that they are cost-effective in comparison with on-demand and reserved instances [2].

Ambient intelligence is the collection of interconnected concealed interfaces that are intelligent enough to satisfy the users' requirements depending on their needs. For some of their tasks, users need some resources. Preemptible resources can be used in the field of ambient intelligence to provide smart environment. In this type of environment, the resources are available based on bidding. One who bids for higher price will get the resource and one who bids lower will lose the resource based on given terms and conditions of the chosen environment. This will create a healthy competition among different service providers as well as among service users. Service providers will be forced by this competition for providing better quality of service. This concept will also force service users for carefully utilizing the services in such challenging environment.

Due to limitation of resources, this concept of preemption can be used to provide competitive service to all and give birth to a bright environment. In this paper, we have surveyed the different ways, cloud providers can utilize their unused resources and cloud users can get the resources from cloud providers in a cost-effective way. Further, various techniques used by researchers are also classified in the article. First, a broad classification is done; then, classification is further refined depending on the techniques used by various researchers. To the best of our knowledge, this is the first effort for categorization of different techniques in the surveyed area.

The main objective of the article can be listed as below:

— Systematically analyze the cloud infrastructures by defining a taxonomy and terminology for SIs.

— Outline existing approaches and their challenges for utilizing unused resources.

[1] The term "spot instance" and "preemptible VM instances" are used interchangeably in this article.

— Provide an overview of different mechanisms to get available resources from cloud environment in a graceful manner.
Present research directions and future challenges in the use of spot instances in cloud.

The remainder of the article is organized as follows: Sect. 2 gives description of basic concepts, taxonomy and terminology for the cloud frameworks. Various techniques for optimal utilization of SIs in cloud are discussed in Sect. 3. In Sect. 4, we cover state-of-the-art challenges and the research directions. Finally, in Sect. 5, we conclude the article.

2 Basic concepts, taxonomy and terminology

This section presents some of the basic concepts and terminology related to spot instances in cloud.

2.1 Distributed computing

In this type of computing, the components of a software system are located at different computers in a network to get better efficiency and performance. Distributed computing studies distributed systems. Distributed system is the collection of interlinked autonomous computers. Each computer in the network has its own memory and peripherals. Computers communicate with each other via message passing in the communication network. For every computer, its own resources are called local, while other's resources are called remote [48].

2.1.1 Development of distributed computing

Early to execute a task with computer, programmer has to submit the job to the computer center and has to wait for the output. But at that time, a significant amount of time consumes in setting up the task, due to which most of the CPU cycles getting wasted. In 1950s and 1960s, various concepts like grouping of similar jobs before processing, buffering, spooling and multiprocessing are introduced to increase the CPU utilizations. But all these concepts do not allow interaction of more than one user with the system and sharing of resources at the same time. In 1960s, the concept of time-sharing system was given, but it had become common in the early 1970s. Time-sharing systems allow various users to execute interactive jobs and sharing of resources. At the same time, enhancement in the hardware technology had resulted into reduction in size and increment in processing power in comparison with predecessors to give concept of minicomputers.

Time-sharing systems allow multiple users to share system resources and accessing systems from remote locations, so it can be said that it was the beginning of distributed computing. Incremental changes to microprocessor technology give birth to workstations which are having same computing power as minicomputers but very less in price in comparison with minicomputers. Parallel advancement in networking technology in late 1960s and early 1970s introduces LAN (local area network) and WAN (wide

area network). LAN is used for communication with in a building, while in WAN computers can be in different cities/countries/continents. Speed of data transfer was limited to 56 Kbps at that time. Growth of networking technology continued and, in 1990s it result into the invention of ATM (asynchronous transfer mode) technology. The ATM technology provides a very-high-speed data transfer up to 1.2 Gbps. Due to this high bandwidth, a new class of distributed applications was introduced which allow the transferring of multimedia applications from one machine to other.

Distributed computing was introduced in the late 1970s by combining computers and networking technologies. Suitable software to exploit the systems and optimally utilize their power is difficult to be available at that time. So, starting from the late 1970s, a convincing amount of research work is being done in the area of distributed computing. However, the field is immature with continuing research works.

2.1.2 Models of distributed computing

Distributed system can be built by using several models. The models can be categorized into five different types [47]—minicomputer, workstation, workstation–server, processor pool and hybrid. All the models are briefly described below:

Minicomputer model As shown in Fig. 1, this model contains few minicomputers which are interconnected by communication links. Multiple users can logged on to a minicomputer simultaneously, a user can access all resources available in network either at local (host), presented on the logged-on minicomputer or remote (guest), presented on the machine other than the logged-on machine. If resource sharing (like sharing of different types of file with each type located on different machines) with remote users is required, this model can be exploited.

Workstation model It is a collection of autonomous workstations interconnected via a network. As shown in Fig. 2, every workstation serves as a single-user machine, so at a time number of workstations remain inactive. Due to high-speed LAN, the resources of inactive workstations can be utilized by the current logged-on user at any workstations.

Workstation–server model As shown in Fig. 3, this model combines the above two models and contains few minicomputers and several workstations connected through a network. Workstations can be disk-full or disk-less. Due to high-speed network, disk-less workstations can utilize the disks of disk-full workstations or can work with minicomputer which works as a server in the network to provide various services.

Processor-pool model In this model, users are connected with terminals having no computing power, but there is a pool of processors that can be used by any logged-on user on the basis of need, as shown in Fig. 4. The pool contains many interconnected microcomputers and minicomputers. The major advantage of this model is that the whole processing power of the network can be used by the currently active users.

Hybrid model Workstation–server model is suitable for building distributed system, and processor-pool model is more useful for the task that involves huge computations. So to utilize the benefit of both, hybrid model, as shown in Fig. 5, combines the above

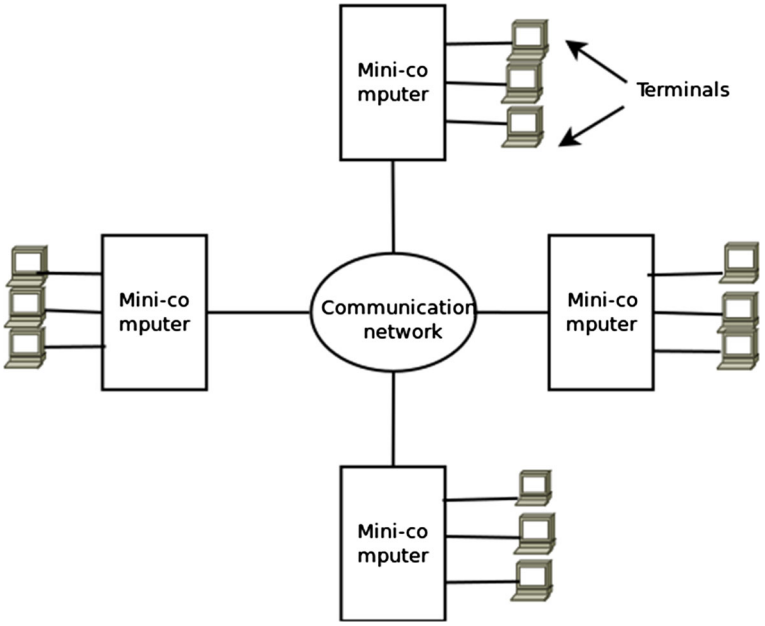

Fig. 1 A distributed system based on the minicomputer model

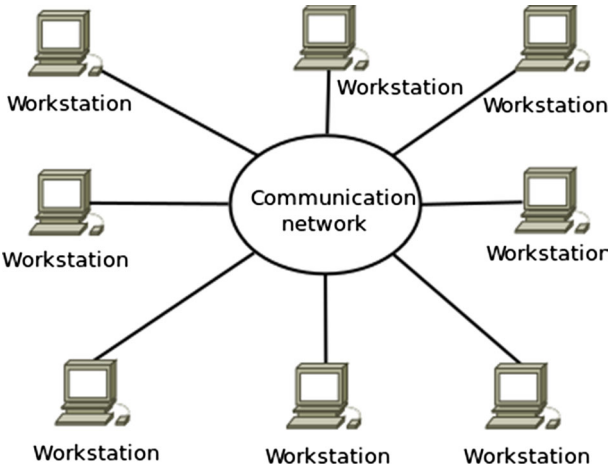

Fig. 2 A distributed system based on the workstation model

two models. The major drawback of this model is that it costs more to be implemented than previous models.

2.2 Cloud computing

The US NIST (National Institute of Standards and Technology) [35] defines cloud computing as "a model that enables ubiquitous, convenient, on-demand network access

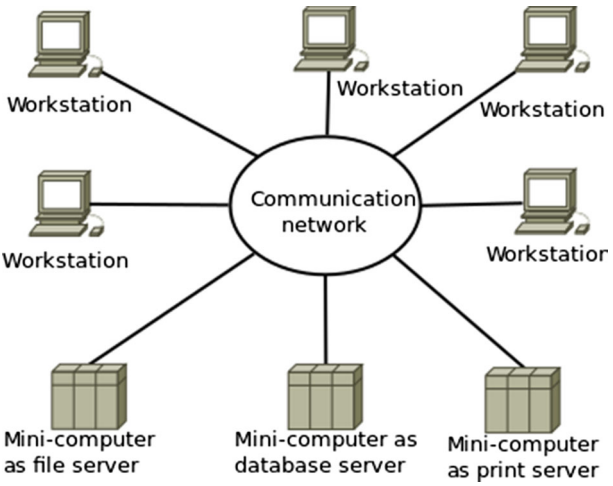

Fig. 3 A distributed system based on the workstation–server model

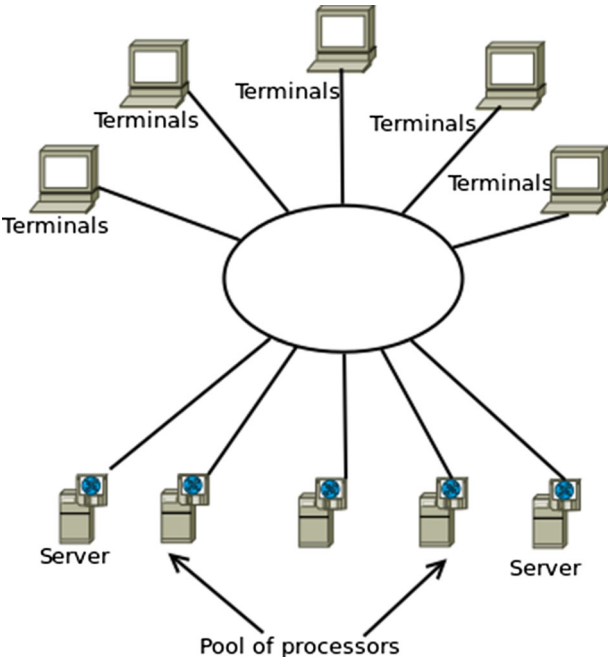

Fig. 4 A distributed system based on the processor-pool model

to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

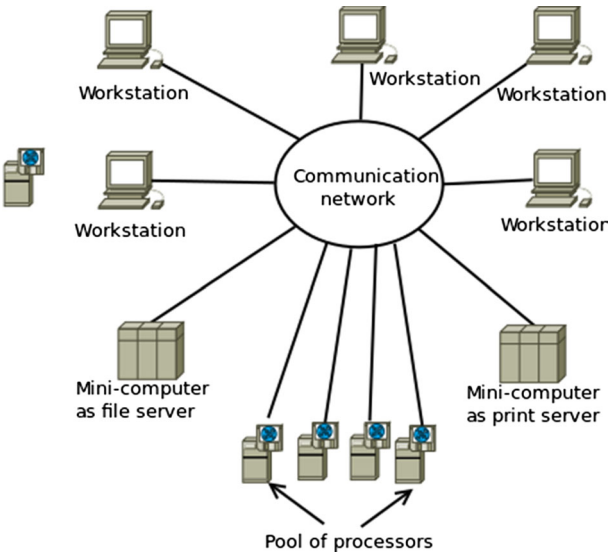

Fig. 5 A distributed system based on the hybrid model

2.2.1 Architecture

The components and subcomponents required for cloud computing form cloud computing architecture. Front- and back-end platforms, cloud-based delivery and network are the components of cloud computing. These components can be organized in the form of layers in which each layer has its own rules and responsibilities. The layered architecture is divided into four layers: user applications, user level middle-ware, core middle-ware and system infrastructure.

System infrastructure layer is implemented by data centers containing several interconnected PCs. Heterogeneous resources, database system and storage services are used to form system infrastructure. System infrastructure is managed by core middle-ware layer which provides a suitable environment for the execution of applications and attempts to utilize resources optimally. This layer also provides virtual version of the hardware to customize runtime environment for an application. This layer is mainly responsible for infrastructure management. Combining system infrastructure with core middle-ware is termed as Infrastructure as a Service (IaaS). User-level middle-ware provides API to develop cloud applications for the users. This layer provides platform instead of infrastructure, so it is known as Platform as a Service (PaaS). First layer, i.e., user application layer, represents services to be delivered at application level. Mostly, they are the web applications that depends on cloud to provide services to the end user. This layer is referred as Software as a Service (SaaS) [1].

2.2.2 Infrastructure as a Service (IaaS)

IaaS provides processing, storage, networks and other computing resources on demand basis to the user. After getting resources, user can install arbitrary software on it. Arbi-

trary software can be an operating system or any other system software. The resources obtained by a user are the virtual resources of the available physical resources. The users have no control on the physical infrastructure, while they have control on the deployed applications on that.

2.2.3 Platform as a Service (PaaS)

PaaS provides platforms for developing and deploying applications in the cloud. It can work as a middle-ware to develop applications with the infrastructure or it provides software to install on the consumer's site. PaaS can reduce the cost of management, deployment and management of deployed applications. The cost of advancement in technology is carried by PaaS providers so PaaS scaled down the risk of enhancement in technology.

2.2.4 Software as a Service (SaaS)

SaaS provides access of users to different applications with the help of Internet. In SaaS, there is no need of installation of software on user's zone and no need of paying to get the license or buy the software; users just have to access the website to input their personal and billing details to use the applications. Different applications like CRM, ERP and social networking can be served by SaaS.

2.3 Virtualizations

Virtualization is a technology that provides abstract environment for the execution of applications. Scalability, flexibility and isolation of execution environment can be attained in cloud through virtualization. This technology combines or divides resources (hardware or software) to get one or more execution environment. An application environment installed on software that emulates devoted hardware is known as a virtual machine (VM).

Virtualization techniques are used to create large number of VMs on single physical machine or hardware. To deal with many VMs in cloud, diverse solutions have been suggested to self-monitor and set up VMs in pool of resources. To utilize spare capacity in cloud, a concept of spot VMs is introduced. Spot VMs are a type of VMs that can be bought by a mechanism of price bids submitted by user in cloud market. Spot VMs offer lower cost in comparison with on-demand and reserved VMs, but they can be withdrawn by the cloud provider as soon as the bid price becomes lower than the spot price.

There are several different types of virtualizations. Some of them are desktop, hardware, network, storage and application virtualization [8].

Desktop virtualization It is used to separate the desktop OS from the system in which it is actually installed. The advantage of such virtualization is that the load can be distributed and one can work in the desktop environment through mobile, laptop or tablet.

Hardware virtualization It is the separation of physical resources from the applications or software which use these resources. The advantage of hardware virtualization is that the same set of hardware can be used for providing different isolated execution environments.

Network virtualization It is the simulation of network hardware resources into corresponding software. It is used to provide virtual network separated from the fundamental hardware. It is controlled by a software-based entity that is a combination of software and hardware network resources to provide functions of a network. Through the concept of network virtualization, one can create single logical network by combining several physical networks.

Storage virtualization Several physical storage of the multiple network storage devices is grouped into a single unit to provide virtualization of storage. It hides the internal functioning of a storage device from the client applications to facilitate storage management without depending on network.

Application virtualization It isolates the application program from the OS in which it is executing. In application virtualization, users feel that application is executing on a local machine, while it is executing on a virtual machine in some other location. The main advantage of application virtualization is that it is less costly in comparison with desktop virtualization.

2.4 Pricing

Pricing is the phenomena of deciding what a cloud provider will get in exchange of services from the cloud user. The services in the clouds are provided at several prices according to different pricing schemes. The ultimate goal of cloud provider is to maximize its profit, and aim of cloud users is to get best QoS at minimum price. Hence, the pricing in the cloud plays a significant role for both service provider and service user. The pricing techniques in cloud can be categorized as fixed-, dynamic- and market-based [45].

In fixed pricing scheme, the charges of services are fixed for longer periods. Fixed pricing scheme charged customers on the use basis that is customers have to pay for the time they use certain services or for the quantity of the product they have used. The cost of the product under this scheme will not change and it is same for every sale or individual. Such pricing schemes are commonly controlled and governed by government.

Dynamic pricing schemes allow to change the price dynamically based on preferences, quantity of purchase and customer characteristics. Service provider will change product prices depending on some hidden algorithms. These algorithms will consider the pricing of competitors, demand of the product and other factors that directly or indirectly affect the need of the product in the market. Dynamic pricing is generally used in business area like retail, electricity and entertainment.

In market-based pricing, price changes frequently according to the current market condition like auctioning, bargaining and demands. In this pricing strategy, service provider will decide the price of the product after analyzing the price of other providers

who are offering the services of similar products. Depending on the number of features offered in the product in comparison with similar product in the market, provider can increase or decrease the product price. In this pricing scheme, need of customers and sensitivity of price is also considered in deciding the price. This type of scheme is commonly used in entertainment industry.

2.5 Spot instance

The unused resources of a vendor are pooled and provided on-demand basis with variable pricing. A user has to bid to get these resources/spot instances. Spot instance is assigned to the user only if the bid price is greater than the spot instance price. If the bid price is equal to the spot price, vendor will decide whether to grant resource or not. A web-based API is provided by vendors for bidding of spot instances. The arguments of the bid request consist of (1) bid amount, (2) number of instance, (3) type of instance and (4) availability zone. The bid request can be categorized into two types based on its persistence property. The two categories are named as one time request and persistence request. In one time, the request is valid for one time, and if the user is not able to get the resource due to low bid, then he/she has to generate a fresh request for the next bidding cycle. However, in persistence, request is active till the time specified in the request. If a user is not able to get the instance in the current bidding cycle, the request is automatically considered in the subsequent cycles till the time specified in the request.

Two more parameters launch group and availability zone for bid request are introduced by Amazon AWS. In a launch group, multiple instances are grouped together to be activated simultaneously, i.e., either all becomes active at a time or none. These types of instances are useful for implementation of big applications or frameworks. In an availability zone, all the instances of a zone can be triggered simultaneously. The spot instances can be terminated either due to bid price becomes less than the spot price or due to unavailability of resources. In both the cases, the partial hours are not charged from users.

Preemptible VM instances or spot instances can be obtained through bidding. Several bidding strategies have been proposed in the literature. For optimal utilizations of spot instances, some authors suggest that either bid maximum price or do not bid at all, while others suggest to bid on-demand price. Depending on their requirement, users can decide their choice of bidding strategy. It is discussed in detail in Sect. 3.4.

3 Literature review

This section provides categorization done by us and state-of-the-art research in the area of optimal utilization of preemptible VM instances in cloud computing.

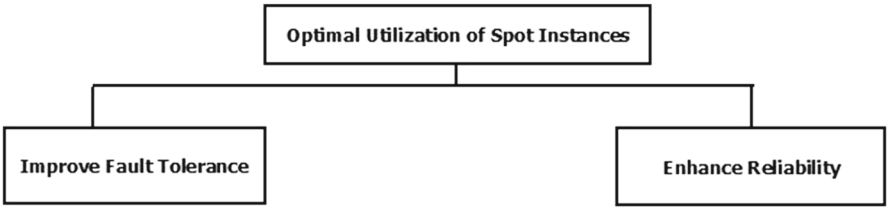

Fig. 6 A broad classification

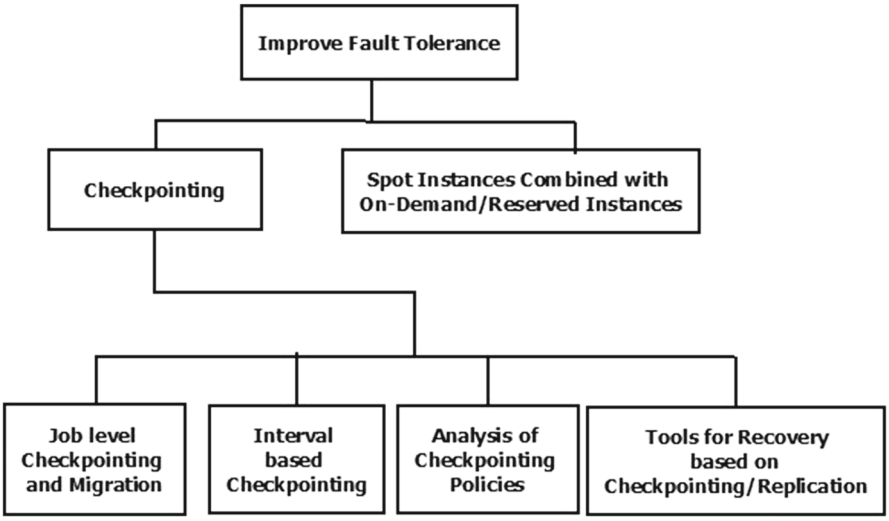

Fig. 7 A detailed classification for improving fault tolerance

3.1 Classification of the various techniques for optimal use of preemptible VM instances

Figure 6 shows the broad classification of the work in the surveyed area.

3.1.1 Introduction

Optimal utilization of the preemptible VM instances can be done by:

- Improving fault tolerance.
- Enhancing the reliability of spot instances.
- Combination of both.

3.1.2 Papers/methods under investigation for improving fault tolerance

Figure 7 shows the representation of work which can help in improving fault tolerance of spot instances.

- Fault tolerance can be improved by:

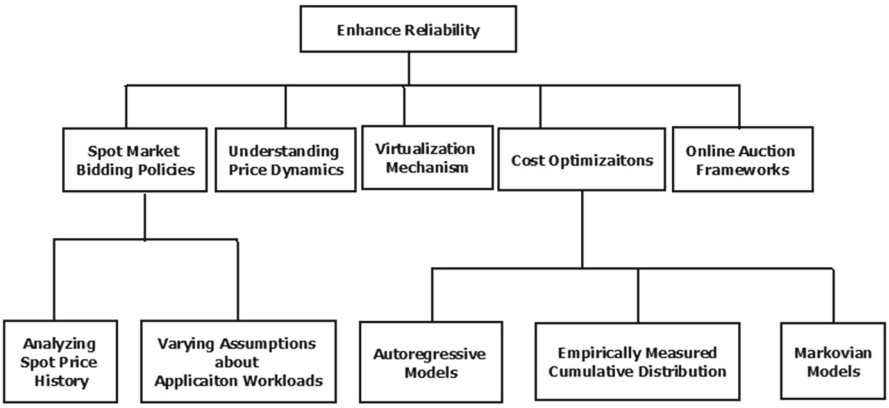

Fig. 8 A detailed classification for enhancing reliability

- Checkpointing
  - Job-level checkpointing and migration [63,64]
  - Interval-based checkpointing [14,26]
  - Analysis of checkpointing policies [30,58,63]
  - Tools for recovery based on checkpointing/replication [14,30,43]
- Spot instances combined with on-demand/reserved instances [17,30,32,36,49,50,54,65]

3.1.3 Papers/methods under investigation for enhancing reliability

Figure 8 shows the representation of work which can help in enhancing reliability of spot instances.

- Reliability can be enhanced by:
  - Spot market bidding policies [18,19,28,34,36,43,49,50,54,65,66,68]
    - Analyzing spot price history [5,23,61]
    - Varying assumptions about application Workloads [33,49,54,65,66]
  - Understanding price dynamics [24,29,46,59,60]
  - Virtualization mechanism [30,58,63]
  - Cost optimizations
    - Autoregressive Models [5,33,59,67]
    - Empirically measured cumulative distribution [17,23,43,44,50]
    - Markovian models [32,36,49,65]
  - Online auction frameworks [6,16,37,40].

3.2 Key performance indicators

The main key performance indicators (KPIs) suitable for common base parameters for methods evaluation and performances for optimal utilization of preemptible VM instances in cloud computing are as below:

– Cost reduction
– Execution time reduction
– Less number of checkpoints
– Minimizing the impact of revocation
– Reducing the number of revocation

3.3 Literature review on improving fault tolerance

Classification of the techniques used for improving the fault tolerance level of spot instances is shown in Table 1.

In Chohan et al. [11], authors analyze the use of spot instances (SIs) to get better performance in MapReduce jobs. The main drawback of SIs is that these are less reliable and likely to be terminated when the bid price becomes less than the spot price. Hence, in case of fault, it will increase the overall completion time. The termination of SIs is dependent on bid price, so the bid price needs to be carefully decided to avoid termination.

Approach Authors use pricing history of Amazon SIs to decide the bid price and number of machines to bid for determining the expected life time of VMs. This tool helps to decide when to use SIs for MapReduce tasks. Since short-running jobs can bear termination, they are the good candidate for SIs. Additional nodes can be used in MapReduce to speed up the execution process; therefore, SIs are good for MapReduce jobs. The number of faults plays a major role in determining the time to complete a process. Market price and maximum bid price decide the termination of SIs. Authors forecast the possibility of termination by using the above information. The authors use Markov Chain and Chapman–Kolmogorov equation to compute the expected life of a VM and thus the quantity of expected work a VM can do. This information can be used to decide when to take backup of data to mitigate the effect of failures.

Experimentation The authors evaluated the proposed approach with two experiments of three applications: word count, pi estimation and sort. First experiment contains five small on-demand instances in which one instance works as the master and four as slaves. The slaves were constructed with two map slots and one reduce slot. In the second experiment, there were five machines as the HDFS cluster and one machine as an accelerator. The authors observe the linear speedup of each job as the number of SIs varies. If the SIs are terminated, it slows down the application adequately. If the SI ran less than one hour, it will charge nothing from user, while if the SI ran more than one hour, it will charge some amount to the user with no work to show for it. The total task time is affected in both the cases. If the HDFS cluster is loaded down with large out going data transfers, the addition of a SI results in slowdown of progress because for each SI, data must reach to it from the cluster. The authors are trying to find this point of shift.

Results/considerations The authors find that in case of faults, all mappers are executed again disregarding whether they will be consumed or not. Checkpointing or saving the intermediate data can be done to avoid re-execution in case of failures.

Table 1 Classification of work for improving fault tolerance level of spot instances

Techniques	Papers	Methods used	Limitations	Possible solutions
Checkpointing	[63,64]	Job-level checkpointing and migration (HBC[2], REC[3], OPT[4], AH[5], AE[6], H+E[7], H+AE[8], AH+E[9], AF(10)[10], AF(30)[11])	Current price is not taken into account Abundant checkpoint resulting into high overhead Migration may result in loss of data and threat to security	Number of checkpoints can be reduced and migration can be avoided
	[14,26]	Interval-based Checkpointing (checkpointing/restart mechanism, over-committed situation, estimated interval-based checkpointing (EIC))	Over-commitment may cause exhaustion of resources which may execute kill events for low priority tasks	Over-commitment may be allowed but with some restriction on the amount of over-commitment
	[30,58,63]	Analysis of checkpointing policies (checkpointing, task duplication, and migration)	Comparison of the pricing and reliability dynamics across markets is not done Cost efficiency is based on resource utilization rather than the operation of provisioned resources The work is not able to find automatically the best cluster configuration for given user constraints on the budget and the availability	Comparison of the pricing and reliability dynamics across markets can be done Design a framework that is able to find automatically the best cluster configuration for given user constraints on the budget and the availability
	[14,30,43]	Tools for recovery based on checkpointing/replication (spotCheck tool using the concept of derivative IaaS)	The risk of SpotCheck having to concurrently migrate all nested VMs at one time is high, since all VMs mapped to a backup server are from single pool SpotCheck VMs cost more than the spot instances	Distribution of backup server can be done so that all nested VMs from a single pool can be distributed Restriction on the number of migration at a time can be done

Table 1 continued

Techniques	Papers	Methods used	Limitations	Possible solutions
Spot instance with on-demand/reserved instances	[17,30,32,36,49,50,54,65]	Spot instances combined with on-demand/reserved instances (profit-aware dynamic bidding algorithm, semi-Markovian process, constraint Markov decision process (CMDP), dual-option strategy)	Single type of spot instances is taken into consideration Dynamic bidding is not taken into consideration Difficulty in the extension to geo-distributed cloud The work does not guarantee a strict deadline	Can be extended to other type of spot instances Dynamic bidding can be taken into consideration Extension to geo-distributed cloud can be done

[2] Hour-boundary checkpointing [63,64]
[3] Rising edge-driven checkpointing [63,64]
[4] The optimal base checkpointing (takes checkpoints just prior to failures) [63,64]
[5] Adaptive hour-boundary checkpointing (decides every hour boundary whether to take or skip) [63,64]
[6] Adaptive rising edge-driven checkpointing (decides every rising edge whether to take or skip) [63,64]
[7] Hour-boundary and rising edge-driven checkpointing [63,64]
[8] Hour-boundary and adaptive rising edge-driven checkpointing [63,64]
[9] Adaptive hour-boundary and rising edge-driven checkpointing [63,64]
[10] Adaptive fine-grained checkpointing (decides every 10 min whether to take or skip) [63,64]
[11] Adaptive fine-grained checkpointing (decides every 30 min whether to take or skip) [63,64]

Paper [63] suggested a way to decrease the cost of a task execution with the use of spot instances.

Approach Authors analyzed various checkpointing mechanisms to minimize the cost and maximize reliability. They used the past prices of Amazon EC2's spot instances to analyze several different dynamic checkpointing schemes which is flexible enough to modify itself based on current instance price to present their advantages in comparison with static, cost-ignorant checkpointing schemes. The main result of the work is that applying dynamic checkpointing schemes results in the significant reduction in financial cost and also increasing the reliability.

Experimentation Implementation is done in the paper by Simulator, that takes the previous spot price history as input to compute the probability density function for rising edge and available durations. This simulator applied 12 different checkpointing schemes on the 42 different spot instances. For each data point, 100 experiments are done to support the results.

Results/considerations Authors observe that in case of failures, rising edge-driven checkpointing scheme (RECS) performs better for some instances, while hour-boundary checkpointing scheme can appropriately decrease the cost. For a small-sized tasks, delayed termination can be used to decrease the costs for a fix task's size of 500 min. There is large performance gap in between optimal checkpointing scheme and other checkpointing schemes. This gap can be reduced by placing checkpoints at the appropriate position.

In Dawoud et al. [12], authors have given a different method named as elastic spot instances (ESIs) approach to solve the problem of abrupt termination of SIs.

Approach In ESIs approach, if the spot price becomes greater than the biding price, instead of abruptly terminating the instances, the service provider scales down the capacity of SIs in proportion of hike in spot price. Due to this, the cost of uncharged partial hours gets eliminated which in turn results in the increase in the profit of service providers. In the ESI approach, the task of SIs termination is assigned to the customer while keeping the control on service provider's side to alleviate the effect of overloaded instances on other services. To handle abrupt termination of SIs, fault-tolerant architectures are required.

Among the four fault-tolerant architectures such as MapReduce, grid, queue-based and checkpointing, the checkpointing is used to handle SIs termination. The benefit of using checkpointing is that it will not require major change in the application of customers, and this technique can be merged with other fault-tolerant techniques. The main benefit of SIs termination by customer is that customer can checkpoint its task before terminating the instances which results in the optimum utilizations of checkpointing scheme. This helps customers by checkpointing the task just before the termination of VMs instance.

Experimentation Xen Credit Scheduler is being used as an accelerator to fix the capacity limit of CPU for the VMs. In work-conservative mode, the scheduler stops the heavy-loaded VMs from utilizing the full capacity of host's CPU which in turn reduces the performance of other VMs. Million operation per second (MOPs) and time of job

execution are noted for the capacity of every CPU. Authors found that there is a linear relationship between throughput and capacity of virtual CPU.

Results/considerations In the said approach, instead of abruptly terminating the instance user can decide to run the task to meet the deadline with increased price. If the spot price becomes maximum price for the user, user can plan for the termination of the instance. During termination of the instance, the user will pay on the basis of per second. The evaluation of the approach is given by modeling the performance of VM instance with various capacities of resources.

In the article [4], it was said that the spot price of VM in Amazon EC2 is not based on market demands; instead, the spot instance prices are decided by using a dynamic algorithm without considering client's bid. Authors show that the high prices are based on market, while the low prices are mostly based on AR(1) (autoregressive) dynamic algorithm. Amazon advertises the spot price while hiding how the price is decided. Recent price history of spot instances is given by Amazon.

Approach Authors analyze the price history to reverse-engineer the method of setting the prices, and a model is built to compute prices which is coherent with the available pricing traces. Client can utilize the proposed mechanisms and policies of pricing to decide best bids, predict the uninterrupted execution time of a spot instance and understand when to buy a low-priced or a high-priced capacity. For providers, the information related to pricing can help them to better exploit their free resources effectively.

Authors claimed that a dynamic algorithm is used by Amazon for fixing a reserve price of the auction without considering the user's bids. Authors also argue that the inputs of the dynamic algorithm are a minimum price and a maximum price for every instance. This range is considered as a pricing band. The change in the reserve price is done randomly by the algorithm so that the relation between prices in the range and availability of instances becomes linear. Availability is a linear function of price. The linear relationship between price and availability ensures that SI's reserve prices lie in the range of minimum and maximum prices. Different minimum and maximum values of price are set by Amazon for different regions, types of SIs and operating systems, thus creating illusion that the prices are based on market demands.

Experimentation A simulator that is controlled by events generated by users. The events can be (1) instance creation and termination and (2) price variation. The grid trace-controlled simulator is executed on 70 CPUs, depending on the count of CPU in the trace. Cloud capacity is defined as the maximum amount of memory that can be used simultaneously in the trace. It is due to the fact of over-commitment of CPU over traces of cloud while physical memory was not committed on traces of cloud. The simulation was terminated on the submission of final job of input trace.

To validate the speculation, authors provide the simulation of the prices and availability. Availability is the result of fixing the auction prices with a reserve price. It is done by setting the number of provided instances which maximizes the provider's profit.

Results/considerations Fixing reserve prices randomly will provide several benefits. A random reserve price can hide the low demand and idle price time, by making an

illusion of demand and supply changes, and so rising the provider's stock. A wider band of minimum and maximum prices can also hide the condition of high demand and low supply and thus provide a false impression of never ending flexible cloud. On the other hand, in narrower band, the use of dynamic algorithm for deciding price within the band will hide the situation of low demands. Authors demonstrated that different price characteristics like minimum value, change timing and band width could vary every six months as the decision taken by Amazon. It is claimed in the paper that arbitrarily employing Amazon's current traces to model the behavior of client on an average 98% of the time become groundless.

Authors analyze checkpointing and migration mechanism for maximizing reliability and minimizing cost & turnaround time of task execution on spot instances [64].

Approach They use Amazon's live price traces to analyze different dynamic checkpointing strategies which can be used with the current spot instance price and present their advantages against static and cost-ignorant schemes. In checkpointing, partial work has been saved at a regular intervals or at some prescribed event, so that in case of out-of-bid event the work from the last checkpoint has to resume in place of the whole work. Various checkpointing schemes like optimal strategy, no checkpointing, hourly checkpointing, rising edge-driven checkpointing, basic adaptive checkpointing-A and current-price-based adaptive checkpointing-C have been described by authors.

In migration technique, user has to bid for another instance type at per-core price instead of waiting for the recently used instances. This will annihilate the waiting time for recovery. Low bid price causes to increase the turnaround time exponentially, due to frequent termination of spot instances. Migration technique decreases the job's turnaround time with minor increase in cost. Migration is based on two factors: first factor is the migrated bid price and second factor is the instance type to migrate. Instance type can be of lowest price (LP), lowest failure rate (LF) and highest failure rate (HF) to utilize free partial hours.

In adaptive checkpointing, it is decided whether a checkpoint is taken or skipped based on the recovery time of the task. A formula has been deduced by the authors to compute the expected recovery time under any probability distributions. The correctness of the formula has been verified for the estimated recovery time whether a checkpoint is skipped or taken. Uncharged partial hour can be utilized at best by bidding less than the mean price which in turn results in the failure of instances at regular intervals. Migration can also be used to utilize these frequent fluctuations to reduce cost of task execution.

Experimentation To evaluate the approach, authors have implemented a simulator. The simulator takes previous history of spot prices as input to compute the probability density function of available duration and rising edges. Authors also model combination of more than 20 checkpointing schemes for 42 different types of instances. To assure confidence in the outcomes, authors have done 100 experiments for every data point. Lower bid price results in the increase in the job's turnaround time due to frequent failure of instances.

Results/considerations Authors have studied the effect of previous traces on the decision of adaptive checkpointing. They observe that smaller window size of past traces

is better to grab the current behavior. However, to model stable statistical expectation, a large window size is required. Authors find that longer window size of past traces is better to decrease the cost in about all bid ranges. Current-price-based adaptive checkpointing-C gives better result than basic adaptive checkpointing-A, if the bid price is equal to or less than mean price. C also provides better performance in case of longer window size of past traces. Likewise A, C gives the same performance, but it is not consistent for changing window sizes. Adaptive checkpointing that considers current price mostly gives best results with the largest reduction in cost. Migration schemes reduces the jobs completion time by more than a factor of 2.5 in comparison with the schemes where it is not exploited with a minor increase in cost. Experiments show that migration technique can decrease the product of cost and time by reducing the job's turnaround time.

Checkpointing in multicore networked systems with reduction in its overhead by exploiting various cores of each system is the main goal of work [21]. In networked system, the checkpointing file can be stored at remote location to avoid failure of local nodes. For large system, bandwidth limitation can be a problem in such schemes of checkpointing. Adaptive incremental checkpointing (AIC) has been suggested for significantly reducing checkpointing file size . The smaller checkpointing file decreases the overhead in checkpointing which in turns results into reduction in job's execution time.

Approach AIC has been designed to exploit free cores for multilevel checkpointing with delta compression (dissimilarity compression between two consecutive checkpoints) to further reduce the size at a required point of time. A free core that is available with every node of live systems is utilized for delta compression and writing result to remote location allowing uninterrupted execution of application in other cores. Authors have categorized checkpoint into two types: local and remote. In local checkpoint, the checkpointing is done on the local disk, while in remote checkpointing, the replicated checkpoint is done at some remote location in the network.

AIC predicts cost by taking stepwise regression and online gradient descent algorithm. It is multilevel checkpointing; first-level checkpointing is done by incremental checkpointing on local disk. Higher-level checkpointing consists of sending files on a slow network by applying page-aligned delta compression between subsequent checkpointing files before transmitting them.

To reduce the overhead of remote checkpointing, AIC observes the page similarity of process and checkpoint at high similarity. AIC also allows simultaneous execution of delta transmission and compression on different cores to avoid interrupted execution of jobs. So, it can be considered as dynamic scheme of checkpointing.

Experimentation To evaluate the scheme, authors develop a multilevel concurrent checkpointing model (Markov model). Markov model is a graph with directed edges. In the graph, node signifies state while edge signifies transition of states. If there is no error, the label of the node/state represents the time elapsed by the task in the state. There are two labels over every edge: first label signifies the probability of transition and second label represents the amount of time task remains in old state before moving to the new state. The performance of AIC is being assessed through six SPEC levels with several systems of different sizes. Markov model is used for showing

the betterment in performance in comparison with various multilevel checkpointing schemes under different circumstances like sharing factor, system sizes and application types.

Results/considerations AIC sufficiently reduces checkpointing file size with minor increase in execution time. AIC cuts the jobs completion time up to maximum 47% in comparison with static schemes with periodic checkpointing.

A just-in-time and adaptive scheduling algorithm has been suggested for scheduling task on the resources available in cloud following spot and on-demand instance pricing models [38].

Approach The suggested algorithm reduces the cost of running applications by an intelligent bidding scheme. The algorithm is not affected by the change in cloud resources having different degree of performance and can tolerate the out-of-bid event of spot instances. Authors have used checkpointing technique to provide fault-tolerant nature.

The advantages of on-demand instance are that user gets the instance till the time of the demand, so this technique provides reliable resources to the user, while spot-instance scheme provides resources at lower cost than on-demand with the decrease in reliability, cost and low QoS (quality of service). Authors exploit the benefit of both the schemes. If users have sufficient time to meet deadline, it is better to go with spot instance. At the same time, if there is short time to meet deadline, go with on-demand instance. A job can utilize both, i.e., initially job can be assigned to spot instance and if the job is not completed and user is running short of time, the job can be switched from spot instance to on-demand instance to be completed within time.

If on-demand price is less than the bid price, the algorithm selects on-demand instance to have high QoS. The algorithm also makes sure that the bid price must be greater than the just previous bid. Failure probability is also considered by algorithm, and if failure probability is greater than the threshold set of user, the algorithm selects on-demand instance instead of spot instance. The algorithm also selects right instance type in both the schemes; in case of on-demand, it chooses cost-efficient instance type for meeting deadline, while for spot instance the algorithm selects the instance with lowest price. The algorithm is having time complexity of $O(n^2)$. Authors' experimental outcomes show that using both the schemes will reduce 70% cost in comparison with schemes where only on-demand strategy is used.

Experimentation To simulate the framework of cloud, CloudSim [9] has been used and extended. Six baseline algorithms, namely ODB (on-demand baseline), SPB (spot baseline), CODB (conservative with on-demand bidding strategy), CNB (conservative with naive bidding strategy), AODB (aggressive with on-demand bidding strategy) and ANB (aggressive with naive bidding strategy), have been formulated to compare authors bidding schemes and algorithms. In the simulated experiment, three levels of deadline (strict, moderate and relaxed) have been considered. For the tasks having strict deadline, high-quality instances are required to complete the task. A moderate deadline task needed high- and low-quality instances required both in combination, while a relaxed deadline can be met with the low-quality instances.

Results/considerations The proposed bidding strategy is compared with the on-demand bidding strategy. Authors find high failures with the naive bidding strategy. The failures will not affect the running time of the task due to flexible nature of the suggested algorithms. The chances of failures in relaxed deadline are more in comparison with chances in strict and moderate. As slack time is high in relaxed deadline, it is low in strict and moderate deadlines. The outcomes of the experiments done by authors depict that every deadline can be met and the suggested algorithms can tolerate failures without regard of bidding strategy. Four different frequencies are taken for checkpointing at every 0 (CHKPT0), 5 (CHKPT5), 15 (CHKPT15) and 30 (CHKPT30) minutes. 0 (CHKPT0) minutes means no checkpointing; in this scenario, authors find 9–14% high cost of executing the task. In spot market, CHKPT30 performs better in comparison with CHKPT5 and CHKPT15 due to less overhead.

Amazon spot instance has been used to design a framework for high- performance computing (HPC) [52]. Authors demonstrated a methodology and designed a toolkit to deal with the challenges of MPI (message passing interface) applications. They devised a formal model and a toolkit named SpotMPI for HPC applications to alleviate the pragmatic execution of live MPI applications on fickle auction-based cloud platforms. The proposed model collects the fundamental dependencies between elements that utilize critical time by taking advantage of bidding histories and arguments of instrumented performance. Authors analyze various algorithms with dissimilar communication vs computing complexities.

Approach Authors exploit auction-based cloud resources to demonstrate a model for forecasting HPC application's execution time. The suggested model considers HPC applications time complexity, checkpoint-restart cost and available past bidding prices. The authors try to resolve the inter-dependencies of communicating/communication complexities of application, the number of processors needed, execution costs and bidding prices. Further, to automate execution of MPI application through exploiting volatile resources of cloud, SpotMPI toolkit has been demonstrated. The suggested model is applied on the current bidding history of the resources of HPC in the Amazon EC2.

Experimentation Evaluation of pragmatic MPI applications on spot instances is done by taking into account the checkpoint-restart (CPR) overhead which is having highest weight in deciding cost. CPR needs to be optimized to get practical acceptance of MPI applications to execute volatile auction-based cloud infrastructures. The devised toolkit (SpotMPI) can observe spot instances and bidding prices for automatically checkpointing at bid price to be managed at optimal intervals and automate restarting of applications on the occurrence of out-of-bid events. The validation of bid-aware CPR interval exploiting non-optimal intervals is done. The behavior of speed up is observed with different time intervals of CPR. The outcome of the authors' experiment shows that when the bid increases the gain of optimal CPR interval decreases. It is because at higher bids, the requirement of periodic checkpointing is about to be eliminated.

Results/considerations The suggested approach provides predictions that can help in the selection of auspicious bidding strategy which is based on critical factor dependencies. The timing model with bid-aware CPR model gives an efficient tool to decide

optimal bid and also the optimal number of required processing units to finish a particular application. This article is the base for other specific pricing models for cloud vendors by giving more in-depth knowledge for the return of investment. The speedup advantage can be compromised if the number of processors attains a threshold. It is useful to provide low price as rebate in volume which is sensible for the complexities of communication. An innovative pricing model can change user conducts which in turn affect cloud providers that finally attain a balance and maximize exploitation of resources.

Authors present a method and a toolkit to face the problems in using unstable resources of auction in cloud for MPI applications in Taifi [51]. The demonstrated approach will be helpful in finding optimal bidding scheme. This work is similar to the research done in the paper [52].

An application-centric checkpointing scheme (ACC) has been devised for application-centric resource provisioning architectures in Khatua and Mukherjee [31]. ACC is used to sufficiently reduce cost and increase reliability in comparison with current checkpointing schemes. To suggest the proposed strategy, authors have analyzed and compared several checkpointing schemes used with spot instances.

Approach In the demonstrated scheme, authors have defined two decision points just before to each hour boundary and follow the three different cases to decide about the checkpoint and termination of instances. (1) If biding price is above the spot instance price at both decision points and hour boundary, ACC will neither take checkpoint nor terminate the instance. (2) If the biding price is less than spot instance price at any decision point but greater than the spot instance price at hour boundary, ACC will take checkpoint but not terminate the instance. (3) However, in case bid price is less than spot price at both decision points as well as hour boundary, ACC will take checkpoint and also terminate spot instance.

Experimentation To evaluate the devised scheme, authors have compared the suggested scheme with the current schemes. The experiments have been done on 64 different type of spot instances.

Results/considerations The outcomes of experiments done by authors show that the demonstrated scheme performs better than the previous checkpointing schemes. This is because ACC permits continuous job execution even if the bid price falls behind the spot price without any effect on the execution cost.

To handle spot instance pricing of multithreaded applications, AIC scheme of Jangjaimon and Tzeng [21] is improved in the article [22]. The modified scheme is named as enhanced adaptive incremental checkpointing (EAIC) scheme. This model considers hardware failure and spot instance retraction for correctly forecasting the point of time to take checkpoint. It reduces the execution time and cost. In the use of spot instance, an application can be terminated due to unavailability of resources caused by either hardware failure or increase in spot price.

If the application is terminated due to hardware failure, the execution can only be resumed from the latest checkpoint when any unit of failed hardware becomes available. If the application termination is caused by price hike, the execution can be resumed from the latest checkpoint, when the spot price of the instance becomes less

than its bid price. To develop adjusted Markov model (AMM), authors exploit exponential distribution of rate of events which are responsible for application termination due to resource unavailability.

Approach EAIC behaves dynamically with the network by estimating network bandwidth and exponential smoothing to obtain improved accuracy. If checkpointing latency increases due to delta compression, EAIC is capable of skipping such type of delta compressions. When EAIC considers few checkpoints to develop its prediction model, this scheme transmits checkpointing file in advance. The file is transmitted in the stepwise regression period at the starting of execution. If the unavailability rate is high or process has short execution time, sufficient checkpoint samples must be transmitted before the development of prediction model.

Experimentation Ten thousand specimens of AMM and model of Yi are simulated through the distribution of these specimens in the duration of 8 months with the Yi's live price traces of 42 different Amazon EC2 instance types. The model is trained with the price traces of a given interval to predict the anticipated turnaround time of the program execution with the error in prediction. This process is repeated for every specimen. The error is computed by reviewing the turnaround time of running the real program. Authors evaluated the performance of EAIC on real scenario and provided comparison of EAIC with its static, basic versions and recent static multilevel checkpointing.

Results/considerations EAIC depicts large metric (cost and time) reduction on application with a higher base execution time which is due to high probability of SI retraction to get more benefit from efficient checkpointing. This reduction gap will be increased for frequently fluctuating pricing instance types. In such scenario, EAIC sufficiently decreases task completion time and cost in comparison with the previous schemes.

An algorithm has been proposed for optimizing cost and time of active simulation on public cloud environment with spot pricing mechanism [27].

Approach The proposed algorithm has been implemented in Python to be used by the active application on Amazon EC2. To measure the cost of executing large applications, the following five factors have been considered by the proposed framework:

1. Time variation between contiguous simulation state and checkpointing events.
2. Capability of bidding various instances and shifting between them.
3. Live effect of instance termination by Amazon.
4. Time lag for instance booting.
5. Correct simulation of Amazon billing mechanism.

The suggested framework can be helpful for live bidding on Amazon EC2.

Experimentation Authors consider two types of simulation problems, namely small and large simulations, to create the demand of high computation time. In small simulation, the time required for execution is small so checkpointing is not required. However, in large simulations, a large number of iteration of executions are needed for simulation having a number of design points or iterative simulation is required for every design point. In large simulations, single simulation executes for several days, so checkpointing is needed to avoid execution from beginning in case of failure of

instances. The number of iterative simulations can be different; generally, it is less than the number of small simulations. Both the types of simulation problems are taken into account in the proposed framework by accurately setting the simulation arguments. Executing small simulation on unstable large servers is cost-effective, while large simulation should be executed on stable and small servers to avoid repetitive checkpointing.

Results/considerations The outcome of the simulation done by authors shows that for both types of simulation problems, the technique of bidding just above the spot price generates good results. This technique is not affected by the reservation price level of various server types and groups until on-demand price remains less than reservation price. The cost does not exceed the minimum reference level by greater than 5% and minimum reference price level is about 17% of on-demand price. Authors also find that free launch mechanism can be used to further reduce costs and the reduction can be about 10% less than the minimum reference price and bid price is equal to spot price. But this scheme sufficiently increases the time about 33% of the time which is at the price of minimum reference price level. The suggested simulation can be used for examining various bidding algorithms suitable for specific projects.

To manage the uncertainty of spot instances, authors have depicted a system named Cumulon [20]. This system is used to write high-level programs involving matrices and linear algebra, without considering the mapping of data and calculations to the cloud platforms. Authors have devised the utilization of Cumulon for efficiently bidding and using extra spot instances to further reduce cost while remaining under the users' level of risk tolerance. They also develop a flexible computation and storage engine for matrices and linear algebra at the top of spot instances. Having an optimized base plan with instances of fixed price, one can enquire Cumulon to involve spot instances in the plan. Incorporating spot instances helps in further reducing costs with meeting users' constraints of maximum cost limit.

Approach Cumulon uses a design of bi-store containing fixed price primary node and variable prices transient nodes called spot instances. To handle the revocation of spot instances, Cumulon utilizes caching as well as sync operation to copy chosen states from transient storage to primary storage. It computes execution time and time required for recovery and sync operations. It has to deal with incoming market prices to predict the spot instance revocation. It transforms the VM's uncertainty to cost uncertainty of its recommendation. Authors call the event of transient node revocation, a hit event. The assumption is that there is not enough time for checkpointing of execution progress on the same node in case of hit event.

Experimentation To optimize bidding, authors assume that in the event of hit, the remaining work will be carried out using only primary nodes. The program will be executed on both primary and transient nodes till the occurrences of hit event; this phase is called prehit phase. If there is hitting and task has not been completed, one enters into recovery phase in which recovery is done by primary nodes to complete the remaining tasks. At last, the completion of the remaining task is done by primary nodes and this phase is called as the wrap-up phase.

To estimate the cost, authors first create a schedule for running both the nodes primary and transient node by taking assumption that hit event does not occur. After

that, authors model various traces of market price. For every trace, they decide the hit time and positioned it in the reference of execution schedule on both the nodes to compute the wrap phase, recovery and finally total cost. Cumulon uses the computed cost of modeled traces, to optimize the likelihood with bounds variability.

Cumulon selects a bidding strategy by performing a grid look-up over all the element pair of bid prices and number of transient nodes. The bid price will start from base price and keep incrementing one cent at a time and number of transient nodes will start from 0 and incrementing by one at a time. The upper bound of the bidding prices is set to the twice of the primary node price, i.e., the fixed instance price. The upper bound for number of transient node is set to 150 as high number of transient nodes result in lessen parallelization returns and larger probability of hitting.

Results/considerations The proposed system reduces cost by exploiting low-cost spot instances with automated cost-based, risk-known execution optimizations, deployment, bidding and syncing mechanism. At the same time, the system maintains the upper bound of the risk which is due to fluctuation in market prices.

A two-layer framework utilizes reliable on-demand instances and low-cost spot instances. The framework decreases the costs of in-memory computer storage loads by using spot instances and also decreases the chance of unavailability with the use of on-demand instances [62]. This framework is named as BOSS (blended on-demand and spot storage).

Approach BOSS rents out on-demand instances that are used to modify and generate objects. These instances are used to process queries that change state. For read-only queries, BOSS uses spot instances.

BOSS rents-out instances from various sites to utilize the varying prices among sites. The effect of sudden termination of spot instances over response time can be reduced by positioning newly created objects at different sites where spot instances are available. This will transfer queries to sites with spot instances. Query rates between sites can be balanced by cloning highly accessed objects to more number of sites than lightly accessed objects. This way of replication decreases the impact of abrupt termination of spot instances; nevertheless, replicating objects at various sites misuses in-memory storage in case of redundant objects.

Authors suggested a different online replication strategy. This strategy takes into account the extra capacity available for further replication and the anticipated cost saving by replicating objects to other sites. To estimate cost, authors analyze the rate of query for new generated objects and history of spot prices. Two components of BOSS are inter-site replication and intra-site replication. In inter-site replication, when a query which generates object reaches BOSS, it uses consistent hashing of Amazon for replication of object to multiple sites [13]. The analyzed query rate with last 24 h spot prices is placed into a cost model with online replication algorithms. This generates the final positioning of data. The objects remain at the final sites till TTL (time to live) expires.

In intra-site replication, BOSS utilizes spot instances for high read request and on-demand instances for high write requests. Distribution of read-only queries is done at many spot instances. These queries are first destined to spot instances, and if the response time becomes greater than twice of the service delay, user can pass queries

to on-demand instances. Users are encouraged to select among available instances for arbitrary reads. For write query of an object to only one site, BOSS uses causal consistency and version number is used for harmonious operation.

Experimentation BOSS is implemented through the use of Cassandra due to its large API. In-memory workload and simple configuration for initiating spot instances are supported by its API. The gossip protocol is also supported by the Cassandra. This protocol helps in identifying and discovering interconnected instances. In Cassandra, the default replication factor is set to 2 days and TTL is set to 7 days. For assuring sufficient storage to preserve all data with default replication, the resetting of reserve ratio is done at the starting of every TTL. The evaluation of BOSS is done at Google and Amazon IaaS platforms. The comparison of BOSS with other schemes is done to prove that cost saving can be achieved with both inter-site and intra-site replication schemes.

Results/considerations The suggested framework costs less than the other generally used substitutes. Comparing with Oracle authors found that wrong forecasting of workload can sufficiently degrade the BOSS's performance. Increased resource utilization can be achieved by better workload forecasting algorithms. The outcome of the experiments done by authors shows that mixing of instances (on-demand and spot instance) can sufficiently decrease the budget of cloud. This decrease can be up to 84% with minor degradation (less than 13%) in performance.

Spot instances of several different configurations along with on-demand instances are used to devise a model for web applications [41]. The proposed model is cost-efficient, reliable and auto-scaling. To compensate the abrupt termination of spot instances, on-demand instances are used with spot instances. The devised system consists of three modules, namely monitoring module, decision-making module and load balancer module. The monitoring module composed of several monitors which are independent in nature.

The aim of these monitors is to retrieve the latest system configuration like rate of request, resource usage, spot prices and status of VMs in the system. Depending on the fetched information, the decision- making module determines the scaling requirement based on the defined schemes and policies when needed. The weighted round-robin algorithm is used by load balancer to divide the request depending on the capacity of every attached VMs. The requirement of the deployed application on the system is that the application should be stateless. Every stateful application can easily be translated into stateless with the help of various means. These means can be stocking the session data in separate cluster of distributed memory caching system.

Approach To handle the abrupt termination of spot instances, scaling system either arranges spot instances of some other type or moves to instances of on-demand type after determining termination. Although the time elapsed in getting and booting a VM is sufficiently large, i.e., 12 min for spot instances and 2 min for on-demand instances, there is a chance that in the meantime another instance gets terminated. To cope up with such scenario, over-provisioning of the application with extra spot instance types is required. Authors define level of fault-tolerant for self-scaling system as the maximum number of spot instances which can be abruptly terminated. This termination should not affect the performance of the proposed system before some provisioning

can be completely regained. Provisioning may decrease the cost saving by the use of spot instances. One possible amendment can be a provisioning of the application by employing greater number of spot instance types. However, the provisions may be more volatile with the involvement of greater types of VMs. The increased risk can be managed by over-provisioned fault-tolerant schemes. Over-provisioning can be decreased by provisioning with mixing of on-demand and spot instances, though this method needs greater monetary cost.

The total capacity provisioned by a single type of spot VMs is known as spot group. Quota can be defined as the VM's capacity that each spot group required to provision. A provision can be secure if the capacity provisioned by every spot group is greater than the quota. The problem of web application scaling with several different types of spot VMs is translated into dynamically choosing spot VM types and provisioning spot and on-demand VMs. This translation is used to maintain the provision in secure state with minimum cost on the increase in application workloads and deprovisioning the VM types, when they are of no use. Scaling system executes spot mode and on-demand mode alternatively.

In case of spot mode, users have to specify the minimum percentage of resources needed for provisioning by on-demand instances. Limit can also be set by the users on the chosen spot groups for provision. The users in an auction always bid maximum price, which is computed dynamically based on the current workload and provision, is called truthful bidding. Three algorithms are given by authors. Algorithm 1 searches for new provision; in case, system needs to scale up, algorithm 2 determines the provision based on the number of on-demand instances and algorithm 3 finds targeted provision if the hour of billing for an on-demand instance is near to complete.

If some instances are terminated abruptly or the existing provision is not able to satisfy the application resource requirement, scaling-up policy is called. In spot mode, the resource requirement means the provision should be secure with the present workload. In on-demand mode, the only requirement is that provision capacity should exceed the need of resources for the present workload. Algorithm 1 is used here to determine the new provision in case system requires scale-up. The instances are billed hourly, so termination of instances will be decided at the end of billing hours. The decision algorithm of on-demand and spot instances is different.

In case of on-demand instances, one not only has to determine the termination of instance but also has to change the spot group if needed. In spot instance, one can terminate the instance if the spot quota can be fulfilled without it. A spot group can be forced out if a spot instance of this type is brought to an end by the provider. The name orphan is given to those instances that are executing without belonging to any group.

Several more optimizations are devised to further reduce cost and increase the reliability of the system. Actual bidding schemes like truthful bidding (bids the maximum price) and on-demand bidding can be used for avoiding out-of-bid event. The system always bids on the price of truthful bid in truthful bidding, whereas in on-demand the system always bids on-demand price of the respective type of spot instance. Revised spot group removal policy says that instead of waiting for provider to terminate some instances of the uneconomical spot groups. But searching dynamically search to find

such group and remove them from provision and find some more economical spot group which have not been chosen yet.

Economical spot groups are those groups whose bid price is less than truthful bidding price. During removal or replacement of some spot groups, orphan in the provision can be utilized instead of provisioning new VMs. The revised scaling-up policy will consider orphan to fulfill the requirement before launching instances to meet the aimed provision. In the revised scaling down policy, spot instance can be immediately terminated, if it belongs to the orphan queue. If the instance is orphan of other spot group, this scheme will terminate the instance. At the same time, some number of spot instances of the type of spot group are started to meet the capacity loss. Since over-provision is in the devised system, resource margin can be decreased.

Experimentation The implementation of the proposed system is done using Java language on Amazon EC2 platform. For evaluation of the performance, a simulation version of the system on CloudSim has been developed. This system is used for comparing the performance of various configurations and schemes by employing traces of real applications and spot markets.

Results/considerations Their experimental outcomes show that the devised approach can sufficiently decrease the resource cost with high QoS (quality of service) in terms of availability and response time.

A flexible just-in-time scheduling algorithm for scientific workflows has been devised by authors [39].

Approach The proposed algorithm appropriately uses spot instances with on-demand instances for decreasing cost, increasing fault tolerance with meeting deadline constraints. The suggested algorithm is claimed to be fault-tolerant against the problem of out-of-bid event and resource failures. Authors exploit redundancy in space and time. A combination of fault-tolerant strategies task-retry and replication are used to reduce makespan and increase utilization of resources with reduced time and cost. For short deadline, task replication is used while for lazy deadline, task resubmission is exploited. To support higher degree of fault tolerance authors replicate tasks on idle slots and on new resources both.

The proposed system consists of work-flow engine which works like a mediator between the user application and cloud. The responsibility of work includes scheduling of submitted task, providing fault tolerance and distributing resources transparently. Work-flow engine contains four components, namely task dispatcher, fault-tolerant strategies, resource allocation and task scheduler. The work of task dispatcher is to analyze the dependencies of data and/or control between tasks and pass on the ready task to the scheduler.

Fault-tolerant strategies like task duplication, resubmission and checkpointing can be used to deal with failures. These failures can be due to software, hardware or any other faults in the surroundings which results in the interruption of task execution. Authors exploited task replication and resubmission as fault-tolerant strategies. The task scheduler selects the type of instance and the pricing model (on-demand or spot). The resource allocation module allocates the suitable resource selected by the task scheduler. The task scheduler utilizes a scheduling algorithm to search an appropriate cloud resources for every task. The suggested fault- tolerant scheduling algorithm is a

type of generalized algorithm. This algorithm can be employed by any system scheduler of work-flow management which supports cloud resources and allow provisioning of spot instances.

The task scheduler component contains four subcomponents, namely runtime estimation, bidding strategies, failure estimator and resource optimizer. Runtime estimation submodule is used to estimate runtime of a work-flow task at a particular type of instance. Failure estimator calculates the failure probability of a specific bid price depending on the history of spot prices. Bidding strategy estimates the bid price. This bid price can be utilized for bidding of a spot instance. The responsibility of resource optimizer is to maximize the utilization of resources.

Authors devised an adaptive, just-in-time heuristic. The heuristic works depends on the place of LTO (Latest Time to On-Demand) time flag corresponding to the current time. The heuristic is explained in four possible scenarios. In scenarios 1 and 2, LTO is after the current time so there is sufficient slack time to finish the work-flow execution within deadline. Spot instances can be used here to further reduce costs. In scenario 1, tasks are mapped to the running instances, while in scenario 2, tasks are mapped to the new spot instances in case of unavailability of running spot instances. In scenarios 3 and 4, LTO is before the current time, so due to short deadline, tasks are replicated to offer fault tolerance because there is zero slack time. Tasks are duplicated either in time or in space depending on the deadline, LTO and the current time.

Experimentation CloudSim is used to model the cloud environment [9]. Authors expanded CloudSim to simulate work-flow applications. It was also extended to simulate Amazon spot market. Spot prices are simulated in CloudSim by using traces of Amazon spot market. Authors analyze the effectiveness of the suggested heuristic in terms of fault tolerance, makespan, cost and resource utilizations.

Results/considerations The outcomes of the experiments done by authors show that devised heuristic creates schedule with sufficiently reduced failure probabilities, makespan and maximize resource utilizations. Authors' experimental outcome proves that the pricing model presented by cloud provider can be utilized to decrease cost and makespan. This model also provides rich and resilient schedules.

Authors in Jung et al. [26] proposed solutions to minimize cost and optimize the performance through checkpointing-based fault tolerance. They provided interval-based checkpointing solutions.

Approach Estimated interval-based checkpointing (EIC) which uses the concepts of weighted moving averages and Bollinger bands is used for increasing the fault-tolerant level. In this scheme, based on past data, upper bound of price and time is determined. Whenever the real price and time surpass this bound, a checkpoint event is generated. Bollinger bands are used to communicate the users about the probable execution cost and time.

Experimentation Simulations were performed on the past data of Amazon EC2's spot instances. The three checkpointing schemes, EIC, HBC and REC, are applied, and their performance is compared depending on bid of user and time of task. The selected instance type for simulation is C1.xlarge (High-CPU). In simulation, interval of user

bid is incremented starting from minimum to maximum bid. Failure probability of the current bid price is also computed depending on the every spot price in the history.

Results/considerations The outcome of the simulation shows that the number of checkpoints required in EIC is decreased in comparison with other checkpointing schemes. Due to adaptive nature of EIC, the rollback time is very less which in turn result in the efficient cost saving irrespective of the spot instance resource type.

A resource allocation policy is demonstrated by authors [58]. This policy is used to reliably execute compute-intensive applications over pools of Amazon EC2 spot instances. Sudden revocation of spot servers is handled by checkpointing application state at coarse intervals. Checkpointing policies are examined for batch jobs to minimize cost of spot instances and mitigate the impact of revocations.

Approach Three approaches are presented to improve fault tolerance: checkpointing, task duplication and migration. In addition to checkpointing and migration techniques, integration of job duplication technique was done. The integration increases the probability that jobs finish within their deadlines.

Experimentation The experiment is designed to analyze the effect of bidding strategy, urgency factor and choice of fault-tolerant mechanisms. The implementation of the simulation is done using CloudSim [9]. The migration, checkpointing and job duplication mechanisms are compared on the metric of cost.

Results/considerations Authors said that the performance of migration scheme will be better when combined with bidding strategies with lower bid values, while checkpointing will be better when paired with higher bid values. However, it was concluded that job duplication yields much higher costs. Comparison of the pricing and reliability dynamics across markets was not made, and spot instances were not supplemented with on-demand instances. Cost efficiency was based on resource utilization rather than the operation of provisioned resources.

A checkpointing scheme on top of an application-centric resource provisioning framework, named as application-centric checkpointing scheme (ACC), has been devised by authors [30]. The scheme increases the execution reliability while reducing the monetary cost significantly. Checkpointing is a common fault-tolerant mechanism for mitigating the impact of revocations on batch jobs in the spot market.

Approach The use of spot resources was encouraged by increasing their reliability in case of outbid event using recovery techniques based on checkpointing or replication. In checkpointing, elastic block storage (EBS) is used for the preservation of finished jobs.

Experimentation A checkpointing simulator is used for the experiment on 64 different spot instance types. The result of the simulation for every instance type is obtained in terms of job completion time, total cost and product of completion time & cost. The performance of ACC with other existing checkpointing schemes is compared.

Results/considerations The result of the simulation reveals that ACC increases cost in comparison with OPT scheme, while there is reduction in cost when ACC compared with other checkpointing schemes. This work is not particular targeting parallel

data-processing engines (PDEs) on spot instances. The work is not able to find automatically the best cluster configuration for a given user constraint on the budget and the availability.

Authors [43] have devised a derivative IaaS cloud platform based on spot instances known as SpotCheck.

Approach SpotCheck is a system that uses nested virtualization and migration mechanisms to manage server pools based on spot and on-demand servers. The platform runs applications in nested VMs, which continuously checkpoint their memory state to a backup server. When notified of an impending revocation, the platform requests an on-demand instance and uses the backup server to migrate the nested VM within the 2-min period between the revocation warning and spot instance termination. To ensure transparency for interactive applications, these migrations must minimize their downtime, which precludes migrating between regions or using local storage.

Experimentation For the implementation of SpotCheck, XenBlanket is changed so that it can support VM migration in bounded time with live migration. In bounded time VM migration, continuous checkpoints ensure that with the last checkpoint nested VM can move the old state within the stipulated warning time. A backup server is used by SpotCheck for checkpointing of VM's state.

Results/considerations Nested VMs are executed by SpotCheck with little declined performance and overhead in cost. SpotCheck wisely uses mixture of spot and on-demand servers to ensure high availability. It approximately provides 99.9989% of availability which is sufficient for all important applications or missions.

A profit-aware dynamic bidding algorithm is proposed by authors [49].
Approach The algorithm observes the current spot price and selects bids adaptively to maximize the average profit of a cloud service broker, while minimizing costs in a spot instance market. The problem was solved using a queue-stability-based formulation.

Experimentation MATLAB is used for simulation. The selected instance is us-east-1.linux.m1.small with the modeling of its spot price variations through semi-Markovian process. Performance of proposed bidding strategy is evaluated using this simulation. Processing speed of spot instance is normalized to one unit job size per hour. The job sizes are created randomly through a bounded Pareto distribution.

Results/considerations The authors claim that the performance of the proposed algorithm is optimal and it provides a generalized framework where other cost objective functions can be adapted. The work does not consider the problem of dynamic bidding to minimize computation cost subject to a finite-time constraint on job completion. Despite successful design in a cloud, this approach has difficulty in extending to geo-distributed clouds.

3.3.1 Conclusion of the literature review on improving fault tolerance

Several techniques like job migration, restarting, duplication and checkpointing (job level or interval based) have been discussed in the literature review for improving the fault tolerance level of task execution on preemptible VM instances. Some authors

analyze several checkpointing schemes to apply checkpointing schemes dynamically for significant reduction in cost of task execution through spot instances. To avoid revocation of spot instances, one approach can be that users can decide to use the spot instance with increase in price to meet sharp deadline. This technique will result into increase in total cost. Some tools or frameworks like Cumulon and BOSS are proposed by authors for increasing the fault-tolerant level of spot instances. Fault tolerance can also be reformed by combining the use of preemptible VM instances with on-demand or reserved VM instances depending on the deadline of the job.

3.4 Literature review on enhancing reliability

Classification of the techniques used for enhancing the reliability of spot instances is presented in Table 2.

The prices of spot instance vary with change of time and space, whereas in on-demand instance user has to pay constant price per unit time utilization of instances [17]. The price of spot instance changes in both the dimensions space and time. So, it is very difficult to forecast the exact future price. For short time, the probabilistic distribution of spot instance prices is static. In temporal variation, the variation in spot price can be large. The change is not fix. The price of spot instance can remain the same for some time and can change abruptly for other time. So, it is too hard to forecast the correct spot instance price, even for very short time.

In the spatial dimension, the change in the spot price for various instance types is not the same. Sometimes, spot price of more strong instance type will be low in comparison with less strong instance type. The change in spot price of an instance type in different availability zones does not remain the same.

Approach The problem of minimizing the monetary cost of MPI-based applications is formulated mathematically, and a numerical method is devised to find near-optimal solution. Optimization overhead can be decreased by three techniques. First is separating the instance type selection from the decisions. Second is modeling the correlation between interval of checkpoint and bid price. Third is a logarithmic search method is being devised by authors for bid price depending on the relation between spot instance failure rate and bid price.

Experimentation The proposed model is applied to NAS Parallel Benchmarks (NPB) kernels version 2.4. On-demand instance is used for application execution with the minimum execution time and denoted it as baseline. Normalization of the cost and time to baseline is done, called as baseline cost and baseline time, respectively. To evaluate the performance, the proposed optimization algorithm is compared with the other existing algorithms, with the use of on-demand and spot instance, and with the other fault- tolerant techniques.

Results/considerations The spot prices are variable; however, the probabilistic distribution of the price of spot instance may be fixed for small time period. Spot price history can be utilized for the probabilistic distribution of spot price for a small duration of time. The change of price in temporal and spatial dimension needs peculiar fault-tolerant design to increase reliability.

Table 2 Classification of the work to enhance reliability of spot instances

Techniques	Papers	Methods used	Limitations	Possible solutions
Spot market bidding policies	[5,23,28,61,68]	Analyzing spot price history (exponential and Pareto distribution, adaptive bidding algorithm, AR(1)[12] dynamic algorithm, fitting a statistical model to the existing prices, Gaussian distribution)	The frequency and time of revocation occurrence are not modeled or predicted	The frequency and time of revocation occurrence can be modeled or predicted
			Long waiting time in the event of underbidding	Long waiting time can be reduced by the use of migration techniques
	[33,49,54,65,66]	Varying assumptions about application workloads (Markov model, base provisioning algorithm, base hard)	The work considers discrete time formulations	Set up can be extended to consider continuous-time formulations
			The checkpoint frequency is not predicted	The prediction of checkpoint frequency can be done
			The workloads with longer running tasks are not included in the article	Workload with longer execution time can be included by extending the work
Understanding price dynamics	[24,46]	Gaussian distributions, descending gradient, linear least square optimization	Due to the complex spot pricing mechanism, the input information is insufficient for decision making regarding bidding	The ranking can be done
			The ranking of cloud services in comparison with on-premises solutions considering both cost and other qualitative attributes, including security is not provided	Comprehensive comparison of the algorithms for the spot price prediction problem can be done
			Users bid is not taken as parameter for characterizing the price dynamics	
			A comparative evaluation of algorithms for the spot price prediction (SPP) problem is not presented	

Table 2 continued

Techniques	Papers	Methods used	Limitations	Possible solutions
			Long-term forecasting is not done	
Virtualization mechanism	[30,58,63]	Bidding lower than mean price, migration, replication	Migration and replication can cause sufficient overhead	The overhead can be reduced by minimizing the number of migrations and replications
Cost optimizations	[4,33,59,67]	Autoregressive models (deterministic resource rental planning, stochastic resource rental planning)	The study does not guarantee strategy proofness	Designing a strategy-proofness technique which also considers time-varying workloads
			Time-varying workloads are not taken into considerations	
	[17,23,43,44,50]	Empirically measured cumulative distribution	Checkpointing and replication both are used so increased cost	Cost can be decreased by using only one technique
			Discarding temporal information	Temporal information can be taken into consideration
	[32,36,49,65]	Markovian Models	Future spot price is only based on the current spot price not the previous spot prices	If the past spot prices are considered, it will give better results
Online auction frameworks	[37,40]	Theory of mechanism design on combinatorial auction (strategy-proof auctions)	Presented without competitive analysis and for a model in which agents cannot misreport their arrival time or patience	Competitive analysis can be done
			Single-valued domains are considered for online real-time scheduling	Multivalued domains can be taken into account
			The approach is not incentive compatible for vector of values preferences	

[12] Autoregressive process of order one

An extensive analysis of all spot instances with respect to spot price and inter-price time at four data centers of Amazon has been done by authors [23]. The inter-price time is the time between two price changes. They have devised a statistical analysis to study the change in spot price patterns at Amazon data centers. They also find the interrelation of time in spot price with respect to hour-in-day and day-of-week. They have analyzed spot price and the time between price change of every spot instance with the blending of Gaussian distribution. A model standardization algorithm has been designed to deal with trend of observed price in the history of real price.

Approach Authors have analyzed hour-in-day and day-of-week time dynamics for the price of dissimilar spot instances at a data center. For hour-in-day time dynamics, they observed increasing pattern in the first half of the day and decreasing pattern during the second half of the day for every spot instances at the data center are observed. For day-of-week time dynamics, they cannot observe any particular pattern of spot prices, excluding decrease in price on the last days of week at eu-west data center. While at other data centers, the accurate pattern in days of the week can be observed. Maximum price on Tuesday, minimum price on Saturday and again increase in price on Sunday are discovered for all spot instances at us-east and ap-southeast data centers. Authors observed that on an average the cost of spot instances may be as less as 44% of on-demand instances. This means that there are some chances of decreasing costs for useful computation at the risk of reliability. However, the highest price of some SIs is more in comparison with corresponding on-demand instances. So, in spite of having high bid equal to on-demand price, the chance of out-of-bid event is still there.

Experimentation Authors have shown by the experiments that ratio between median and mean for inter-price and spot price time of SIs is near to 1 for every trace. This shows that Gaussian distribution might be a good choice for the modeling. However, values of skewness and kurtosis prove that the fundamental distributions are minor-tailed and right-skewed. So, Gaussian distribution might not be a representative model to be used and an improved distribution is in order. Authors find that inter-price time is more variable in comparison with spot price because of high values of variability coefficient. For better modeling, authors have focused on sequential development of spot price and inter-price time. For this, authors analyzed the spread plot of spot price and the time between price change for the duration of February 2010 to November 2010.

Results/considerations To validate the proposed approach, authors have simulated model in CloudSim [9]. The outcome of the experiments done by authors shows that model forecast the total cost of job execution on spot instances with a high degree of correctness. The demonstrated model would be beneficial for spot instance users and educators in Amazon EC2.

Spot instance pricing as a Service (SipaaS) framework, which is an open source, is given by authors to implement dynamic prices for IaaS [56]. SipaaS is like Amazon EC2 for web services to price auction and allocation of spot instances on the basis of RESTful services set. Users have to install add-ons in current platform to utilize the framework. OpenStack platform has been extended to utilize web services of SipaaS for running spot market. It offers services for pricing VM instances through interior pricing module depending on auction scheme of online Ex-CORE pricing

mechanism. Any new pricing mechanism can be implemented by enhancing SipaaS architecture without changing the architecture of web services. Researchers can exploit this framework for carrying research on dynamic pricing schemes.

SipaaS facilitates services to add, remove or modify orders of bidders for different types of spot instances for each cloud provider and calculates the cost for each type. Every VM type of a cloud provider is treated as a separate spot market. It also calculates price of each market on the basis of orders submitted for that type of VMs. The suggested framework is agnostic for the cloud platform and resource management system utilized by providers of cloud. To utilize the services of SipaaS, cloud providers are required to implement their extensions.

Approach The proposed framework consists of two major parts: SipaaS framework and Platform extensions of cloud providers. SipaaS framework contains set of RESTful services that is written in Java and deployed on a server to offer web services needed for functioning of spot market. This framework consists of three primary components: pricing module, database and web services. Pricing module is responsible for setting the price of spot market. It takes list of orders with the reserve price and the free VMs to be set by the provider to calculate the price of spot market according to that. Data related to each spot market is organized in the form of relational database.

SipaaS consists of set of web services for supporting the functioning of spot markets. Cloud provider's platform extensions are a supplementary software component to be installed on a module of cloud provider's platform for exploiting the web services of SipaaS.

Experimentation To assess the performance of the suggested framework, the authors evaluated the framework in twofold. A practical study with a 10-participant group is performed by authors to validate and examined the suggested framework. Authors also examined the behavior of users who participate in the study. They measure the scalability of framework at high loads of service.

Results/considerations The outcome of the validation done by authors shows that cloud providers who want to run spot market like Amazon EC2 can utilize the suggested SipaaS framework. This framework exploits the Ex-CORE auction algorithm as an admissible substitute.

Optimization of cost for applications based on message passing interface (MPI) with the restriction of time limit is developed on Amazon EC2 [17].

Approach Authors mathematically formulate the optimization problem by using both spot instance and on-demand instance for the assurance of meeting deadlines. A numerical method has been devised to find nearly optimal solution. Since spot instance can be terminated at any time when the spot price becomes higher than the bid price, a fault-tolerant execution is needed. Checkpointing and replications are two generally used fault-tolerant techniques. Authors have analyzed these two techniques for minimizing cost of MPI application at the performance requirements specified by users. Authors observed that these two schemes are complementary to each other. To minimize the cost of failure, checkpointing can be used and to decrease the chance of failure, replication scheme can be beneficial. To combine replication and checkpointing schemes

on Amazon EC2, one need to make three decisions (S1) instance type (S2) bid price (S3) time to checkpoint.

The solution space of the mathematical problem is wide. So, authors devised three schemes to decrease the overhead of optimization. First, authors dissociate the decision (S1) from the other two decisions S2 and S3. If the solution space for S1 is N and for S2, S3 is M. By using divide-and-conquer approach, authors can reduce the whole solution space from $N \times M$ to $N + M$. Second, to reduce the optimization overhead, authors have analyzed the relationship between checkpoint interval and bid price. Third, authors have presented a search method based on logarithm which depends on the relation between failure rate of spot instance and bidding price. The demonstrated method can sufficiently reduce the overhead of optimization with quality search results.

The two inputs of the suggested algorithm are the analyzed results of MPI applications and history of spot prices. Authors first select the on-demand instance type and passed it as input to the bi-level optimization algorithm to determine the price of optimal bid and checkpoint interval. In the two-level optimization algorithm, first level is to find the relation between bid price and checkpoint interval for reducing overhead of dimension and also design checkpoint interval as a function of bid price. In the second level, authors use logarithmic search method to find the optimal bid price.

Experimentation Authors implemented the suggested model at Amazon EC2 to analyze its performance with NAS Parallel Benchmarks (NPB) and one real application, LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator). The outcome of the experiments done by authors presents that between two fault-tolerant schemes, checkpoint and replication, one is selected adaptively because of dynamically changing spot prices.

Results/considerations The end results show that the demonstrated approach can decrease cost up to 20% in comparison with state-of-the-art algorithms with meeting time-limit constraint.

An auction model has been suggested for Amazon EC2 to determine price of spot instance by authors [7]. Authors examine the strategies of bidding associated with the model. They also analyze the Amazon EC2's instance pricing. To examine some of the concealed prospect of Amazon's auction mechanism and to search potential profit from untruthful bidding, history of spot instance prices is used by authors. Authors find strategic prospect of communication between cloud user and providers in the Amazon web services setting.

Approach A single instance or launch group user should set bid price same as he/she wishes to pay. If user bid more than his wish to pay, he/she will get nothing when the spot price becomes lower than the bid. It may be that the instance will set at a price more than his/her wish to pay which will result in loss. If user bid less than his/her wish to pay, he/she pay the same amount when the spot price is less than the bid price, but it may be that the price will become more than the bid, but less than he wishes to pay; in this case, user can gain by high bid. Multiple instances or launch groups users should bid the same as they wish to pay for first bid and for other following bids users should bid less than the maximum they wish to pay.

Experimentation Authors devised an experiment that is able to disclose several prospects of Amazon EC2 schemes. Authors wrote a script in MATLAB to calculate the optimal bids for various values starting from lowest to highest prices noted in the data.

Results/considerations The outcome of the experiment done by authors shows that one cannot ignore the effect of bidding at low price and that a user should not bid at a price of maximum, he wishes to pay. The demonstrated experiment of strategic bidding gives a method which exactly disclose the potential profit from a user who diverts from true bidding.

An optimum bidding mechanism has been demonstrated by authors for spot instances of Amazon EC2 [53]. The strategy is named as AMAZING that exploits both the dynamic pricing model and past data. This strategy is used to maintain balance between reliability and costs of unstable resources like spot instances. AMAZING utilizes the intellect of state transitions among numerous spot instances to alleviate in decision taken at the time of job's execution. The decision-making problem is modeled as constraint Markov decision process (CMDP). Utilizing this model, one can get an optimum randomized bidding mechanism using linear programming. AMAZING employs decision of optimum bid at each instance hour till the completion of job execution.

Approach The authors demonstrated two simple and effective heuristic bidding mechanisms with high reliability and low costs in general. First is a heuristic, i.e., Always Bidding Maximum Price (ABMP) in which one always bids the maximum price irrespective of the present state. ABMP assures continuous execution of any task to minimize the time of job completion. In the present setting of spot instance, the maximum price is at maximum 1.1 times of the minimum price, so ABMP will cost at maximum 10% of the minimum cost. ABMP assures lowest completion time and at maximum 10% high of the minimum price. Second is AMAZING, i.e., CMDP-based Dual-Option Bid Strategy, which exploits both intellect of state transition and dynamic pricing simulation. Price transition probability matrix (PTPM), background system which executes a linear programming routine to find a bid choice. The PTPM of a specific spot instance may be obtained from the history of spot prices. One can develop EXE PTPM matrix with E various spot prices of a specific type of instance, to show the probability of transition between pair of each spot price. Any bidding mechanism can be modeled with dual-option strategy in which the dual option is either bid at maximum price or give up.

Experimentation To evaluate the model, authors have simulated the bid optimization algorithms depending on the live price traces with respect to total price, task completion time and the overhead of restart and checkpointing. The assumption is that checkpointing cost of executing program is known and task is running on a single instance only. Authors analyzed how the limiting factors like budgetary constraint affect the distributions of the running time and monetary cost per instance. They studied the overhead of restart/checkpointing at the time of job's execution.

Results/considerations They noticed that the gain of AMAZING is more clear when length of the task increases. Authors' experimental outcome shows that AMAZING performs better than past works with respect to both running time and monetary cost.

The optimal bidding strategy AMAZING has been revised by taking into account changing prices and brilliance of transition among various VMs [55]. In the devised system, a broker is prepared for retrieving the price pattern in the dynamically changing spot prices. This broker offers STPM (spot price transition probability matrix) for probability of spot price transition of a particular type of spot instance. The bidding process is modeled as a CMDP (constrained Markov decision process). AMAZING utilizes the decision of optimal bid at every hour during job execution after getting the solution of CMDP.

Approach An optimal bidding mechanism that minimizes the monetary cost with satisfying deadline constraints can be attained by selecting from the two choices which are either bidding maximum or no bid at all. At each instance hour, AMAZING algorithm precalculates the bid choice probability for the next instance hour (bid or no bid) depending on the present-state information of the instance. Based on the decision, it can be determined whether there has to be checkpoint or not. Three bidding strategies, namely (1) always bid at on-demand price, (2) adaptively setting backup point-valley bidding and (3) CMDP-based bidding strategy, have been proposed.

In the first strategy, one always bids at maximum and it is assumed that the maximum price will never cross the price of on-demand instance. If this is so, then it is better to go for on-demand instance. If the present spot price is less than the price of bidding, one has to pay based on the present price irrespective of one's bid.

In the second strategy, the checkpoint or backup point is set periodically or fine-grained checkpointing is performed. In periodic checkpointing, the tasks with their progress are to be checkpointed at every hour boundary. While fine-grained checkpointing assess whether checkpoint is taken periodically at the interval of 10 or 30 min or not. Valley bidding strategy suggests users to bid maximum when the price of spot instance falls down. In the best case, the users will have to pay lowest price for the instance hours attained by them. While in the worst case when there is increase in price, the users must give up after checkpointing the intermediate results.

In the third strategy, a decision regarding bid is taken at each instance hour depending on STPM for the incoming hours. It also finds the suitable time for checkpointing in advance. A background system executes a linear programming (LP) function depending on the collected information of state transitions in the STPM to determine the bid option. The STPM of a specific instance can be educated from the history of spot price available on AMAZON EC2. Spot price changes in a small range over a comparatively long period.

Experimentation To evaluate the strategy, authors verified the experimental results and observe that AMAZING scheme exceeds the hour strategy. With the increase in the length of task, the benefits of AMAZING become more clear. With the same task length, cost incurred in AMAZING scheme is very less than HOUR strategy for different bid prices. The information obtained from STPM for state transitions will be used by AMAZING to get low price spot instances by EC2 with utilization rate keeping high and meeting deadline constraints.

Results/considerations Authors' experimental outcomes show that HOUR strategy with different bid prices is not able to satisfy the budgetary limitation of 0.18 USD while AMAZING scheme can meet the budgetary confidence greater than 15% with strict cost constraints. Thus, AMAZING strategy exceeds HOUR strategy with respect to execution reliability as well as monetary cost.

A novel algorithm has been proposed by authors for forecasting the Amazon spot instance price [46].

Approach To predict spot instance price, authors have used history of spot instance prices from August 2013 to April 2014. The collected data consist of changes in all various types of spot instance's price at several AWS data centers. The rough data is prepared by translating irregular data into regular time series. And taking average of data at a regular interval of one hour to decrease the amount of calculation needed for more analysis and processing. The assumptions of the algorithm are that spot price is affected by principally two factors. First is global tendency captured at monthly or daily basis. Second is local trend which depends on abrupt increase or decrease in the need of a specific spot instance at a time. Since Amazon spot instance price changes at the completion of an hour, the proposed algorithm predicts the price at an hourly basis.

Experimentation Proposed algorithm has been evaluated based on two categories, namely short-term prediction and long-term prediction. In short-term prediction, the price is predicted for the next hours by taking the data up to the last hour just before the prediction hour. This scheme is useful for the adaptive bidding in which users change their bid price at hourly interval. Long-term prediction is used to predict the spot price before one week. Long executing program or application can use this scheme. This scheme is itself divided into two categories based on day and week ahead. One-day-ahead scheme predicts price for next 24 h of a day by using data till the day before prediction, while one-week-ahead scheme forecasts price for the next week by taking data till one week before prediction.

Results/considerations The outcomes of the simulation performed by authors show that the proposed algorithm can predict spot price one day ahead with 9.4% error rate and 5 days ahead with at most 20% error rate.

A temporal and spatial trend analysis of ten different optimized spot instances with respect to low price hours and weekdays has been devised by authors [57].

Approach Authors have collected history of spot prices starting from April 2015 to March 2016 from Amazon EC2 console. The history comprises of approximately 9,200,000 records. The collected data is sufficiently large to find the dependency of spot instance pricing for ten various optimized types of instance in the nine regions of Amazon EC2.

For hourly drift, authors observed that minimum price period varies differently for different regions but approximately the same for both Linux/UNIX and Windows instances. In comparison with Windows instances, Linux/UNIX instances have more changes in spot prices at different hours of time. However, for weekdays drift, authors find approximately same minimum price days for different regions and they are Sundays and Saturdays. Mondays also have lower price at some regions. The chances of

finding minimum price on Sundays are 57%, while on Saturdays it is low, i.e., 37%. This probability is reduced to 43 and 32%, respectively, for Windows instances. Other weekdays can be considered as high price days.

Experimentation By taking sample of spot price history, authors also found the correlation coefficient between hourly and weekday price trend among various types of instances of a particular region. A hypothetical test of the importance of correlation coefficient is to determine whether linear statistical relationship in the history of spot prices for various instance types is sufficiently strong. So that it can be used to model relationship in prices of spot instances. This type of data is beneficial in deciding how price of one spot instance type varies irrespective to other type of instances in any area of Amazon EC2.

Results/considerations An intermediate to strict linear relationship has been found between various types of instances of a specific area in whole hours of a day and in every days of a week. The reason of this is that, for both hourly and weekly drifts, coefficient of correlation is near to 1 and sufficiently dissimilar from 0 for instance types of Linux/UNIX.

A Q-learning-based bidding strategy has been devised by authors with concentration on execution time, workload, previous checkpoints, past noticed spot instance prices [3].

Approach Authors have taken an application provider to run periodic long running batch jobs on the VMs provided by an IaaS provider. The provider makes sure minimum throughput with QoS constraints. IaaS provider presents on-demand and spot VM on the pay per-use basis to the application provider. To deal with unreliability of spot VM, periodic checkpointing scheme has been used. The aim of application provider is to reduce costs with meeting QoS constraints. So, provider has to determine optimal VMs type (on-demand and spot VM) to be requested and in case of spot VM the amount of bids to be submitted.

The assumption of authors is that application provider can only determine the bidding scheme with the experience. As application provider is not aware of the behavior of other IaaS users, their number, forthcoming spot prices and also with the current spot price history. Authors formalize the bidding scheme as a Q-learning problem. They first model the system by taking only single periodic job to run on single VM. Further, it is shown that same model can be generalized by considering multiple jobs to be executed on multiple VMs at a time. The demonstrated Q-learning bidding algorithm can be effectively implemented. This implementation is used to find the choice of action when there is no slack time; in this scenario, the only choice of VM is on-demand instance. If there is some slack time, user bids for a spot instance and require to compare whether it is good to bid maximum value or zero value, i.e., not to bid at all.

Experimentation The evaluation of the devised strategy is done under various scenarios and live spot price traces. Authors study the behavior of application providers under different IaaS pricing policy with various frequencies of checkpointing. Authors compare their policy with other policies which is based on random request of on-demand and spot VMs and also with Markov decision process (MDP) strategy.

Results/considerations The outcomes of the experiments done by authors show that by using Q-learning, the application provider confines its behavior periodically to decide the action. This action minimizes the cost and maximizes the profit, with better results in comparison with blind mechanism and MDP strategy.

The concept of super-cloud has been used to optimize price and availability of VMs [25]. To surmount the incapability of forecasting incoming prices, authors analyze VMs scheduling in the spot market of Amazon EC2 through live VM migration. Smart spot instances are devised by authors for super-cloud.

Approach With the help of live user-level VMs migration, smart spot instances can attain high availability with reduced cost. Two layers of VMs are used for nested virtualization. First-layer VMs (fVMs) are Amazon spot instance and nested- or second-layer VMs (sVMs) are user VMs. Nested virtualization permits migration of sVMs following a scheduling mechanism. The super-cloud observes the spot price as well as real migration of sVMs to perhaps less costly fVMs of either other spot instance type or spot instances in other region, zone or cloud. At the worst, sVMs can be migrated to regular on-demand instances which are always available. So, the price can be as much as on-demand instance price.

Various types of fVMs can grasp dissimilar number of sVMs. Placing sVMs on fVMs is the main challenge of smart spot instance scheduler. The price of fVM changes periodically, and there is sufficient price difference between various types of fVMs. So, high cost saving can be attained by migrating sVMs between various types. The authors devised a centralized scheduler to optimize the placement of sVM groups. The current placing of sVMs in the super-cloud is recorded by scheduler. The scheduler observes the spot price and determine the fVMs types and placing sVMs at the hour boundary of fVMs. Authors considered only a single type of sVMs, i.e., all sVMs have same configuration.

The aim of scheduling algorithm is to obtain the cost-effective way of placing sVMs on the completing fVMs. The scheduler can either continue to work with the existing group of completing fVMs or allocate new instance types of fVMs. The scheduler also permits some or all completing fVMs to terminate. Before terminating fVMs, the scheduler has to move all the sVMs outside. Two scheduling algorithms, namely greedy and dynamic, have been used by authors. Greedy algorithm is simple and in this single type of fVMs is used at the time of scheduling which results in simple algorithm. Dynamic algorithm attains a local optimal cost and placement. Dynamic programming algorithm is not as scalable as greedy, but probably it can achieve lower costs. To increase the reliability of the spot instance, replication mechanism has been used by authors.

Authors looked into several design choices and used dynamic programming scheme to show some introductory outcomes of experimentation in virtual as well as real environment.

Experimentation For simulation, authors have taken a pair of 10-day price history received from Amazon. Three categories of grouping strategies, dynamic, static spot, static regular, have been compared with margin time of 5 min. In dynamic scheme, dynamic programming and greedy algorithms are used for dynamically placing and migrating sVMs. Static spot scheme retains with starting placement of sVMs. The

starting placement can be either selected from sole spot instance type or best of several spot instance types combinations following the spot prices at the starting of periods. A type of regular on-demand instance is employed in static regular schemes.

Results/considerations They have shown by the experiment that a sufficient amount of cost saving can be achieved by migration in comparison with combination or single type of instance.

Authors in Zheng et al. [68] investigated spot instances in not only the biding strategy of cloud users but also the pricing strategy for the provider. Rather than exploiting existing pricing schemes of cloud services, this work aims to design a new type of service which is resource efficient and cost-effective for delay-tolerant jobs with deadlines. Further, they examined various bidding policies to determine the optimal bidding strategy to minimize costs, while ensuring an application either finishes within deadline or maintains a specific availability target (with high probability).

Approach This work uses exponential and Pareto distributions to accurately model the probability of the spot price equals a particular value in selected markets. These models are useful in predicting availability at a given bid price. They do not model or predict when and how often revocations occur. In this case, when the bid price is below the spot price, applications simply wait until the spot price falls below the bid price again before resuming execution. This lengthens the running time of batch jobs and decreases the availability of online services. Dynamic pricing plans from multiple cloud providers can be explored to improve cost efficiency.

Experimentation Implementation of the proposed bidding strategy is done to execute Hadoop MapReduce jobs over the Common Crawl Dataset by using Amazon Elastic MapReduce (EMR). Comparison of the job completion time and cost of executing MapReduce jobs over on-demand as well as spot instances is done for the five different settings of client. On-demand instance is considered as a baseline to show that MapReduce jobs' bidding strategy significantly reduces cost with a slight increase in time.

Results/considerations Bidding for multiple spot instance can lower time as well as cost while if one uses on-demand instance, it will result into increase in cost with decrease in time. The proposed bidding strategy can reduce cost by approximately 90% through the use of several instance types with a minor increase in time.

Authors developed optimal bidding strategies in spot markets both from a client and from a broker perspective [65].

Approach From the client's perspective, authors designed a dynamic bidding policy (DBP) to minimize the total cost of a parallel or serial job with a deadline constraint. They study dynamic bidding policy for spot instances in the context of deadline constrained jobs. They employed Markov chain models to characterize the spot prices.

Experimentation Finite-time stochastic dynamic program is used in the formulation of problem for minimizing the computation cost. First a class of parallel job is taken into consideration to find bounded analytical result for a function which is recursive. Optimal bid is calculated by this recursive function. Graphical representations are used to retrieve several new hidden concepts from it. The enhancement of the result

is done to incorporate class of serial jobs where serial computation is done in which maximum single unit of computation is done per unit time. At last, the performance of the proposed algorithm for several VMs with different configurations is analyzed using the real data set of Amazon EC2.

Results/considerations Analytical and bounded result for the optimal strategy within a Markov spot price evolution was achieved by authors. They do not predict optimal checkpoint frequency.

Authors presented a provisioning algorithm for extending cluster capacity with spot instances [33]. The goal of such algorithm is reducing waiting time of jobs in cluster queues, where users have reservations that define amount of time required by the job and number of resources required by the job. They proposed dynamic policies to purchase the less expensive spot instances to satisfy the peak demands for computing clusters. A heuristic algorithm was proposed for additionally provisioning servers (e.g., spot instances) to accommodate workload in a local cluster.

Approach They considered the economics of purchasing resources on the spot market to deal with unexpected load spikes. Two scheduling policies were offered to control the arrangement of virtual resources and the time when they will be free. The policies change its behavior dynamically with the variation in the task queue.

Experimentation A discrete-event simulator has been developed, and some performance metrics have been identified to analyze the performance of experimental results. In the simulator, resource boot time is of 3 min and maximum 200 external resources can be used at a time.

Results/considerations Many times the fluctuation of the spot price is done with in a fix range. Hence, a small change in price will result in big effect. Cost can be low with larger bid price, as major work is performed on the resource of spot type not on full priced. The approach does not take into account the workload type and the resource failures to make a decision about the redirection of requests. These models are no longer relevant once a drastic policy change is made.

A resource rental planning with EC2 spot price predictions to reduce the operational cost of cloud applications has been demonstrated by authors [67].

Approach They assumed that spot instance prices are market-driven and modeled some of them to be used as a client decision aid. They leveraged EC2 spot price prediction to design a resource renting strategy to reduce the cost of cloud applications.

Experimentation Three VM classes, c1.medium, m1.large, and m1.xlarge, are taken for two set of simulations to analyze the performance of proposed resource rental planning strategies. The decision of rental planning is done on the basis of hour. The assumption is that required software by application services has been installed on VMs which is taken from cloud on rent basis. Hence, the starting process to set up the VMs has not been considered.

Results/considerations The price of spot instance cannot be correctly approximated to be utilized in the model of deterministic type. The devised model of optimization provides impressive medium for resource rental planning.

Authors used empirical cumulative distribution function (CDF) of historical spot prices for spot price prediction [44].

Approach The CDF-based models end up discarding valuable temporal information about the continuity of the spot price staying below different bid values, which may lead to poor decision making. Users should concentrate on buying economy EC2 servers through the use of easy bidding mechanism and neglect the difficulties in bidding. In the use of changing price resources, the applications must be changed to manage sudden revocation and allotment of resources. At the occurrence of revocation event, application must be moved to resources of least cost and having excellent performance.

Experimentation The correlations of price change between markets are analyzed by studying past prices of a single spot server across different availability zones of a region and also across different server types in a single availability zone.

Results/considerations The authors claim that complex bidding strategies are not beneficial in comparison with ordinary strategies. It is due to many factors. First, price characteristics provide vast range of bid prices that result in optimal availability and cost. Second, there is always availability of markets with extra resources due to the large count of spot markets. Hence, user can easily migrate for new resource in some other market when the price of current resource becomes high. Third, one can bid high price without any surcharge.

Bidding and checkpointing policies were examined in Menache et al. [36] for batch jobs to minimize the cost of spot instances and mitigate the impact of revocations.

Approach An online learning algorithm has been proposed by authors for resource allocation and to handle the trade-off between performance and cost. The concept of machine learning is used to design the algorithm. The objective of the machine learning is to learn policies with good performance with the given candidate policies. The proposed algorithm manages the bounded delay and provides a formal guaranteed performance. The performance changes with the change in the amount of delay and the number of jobs waiting for processing.

Experimentation The performance of the learning is evaluated through simulation on artificial data and real dataset collected from a large batch of computing clusters. The advantage of using artificial data set is that it permits the flexibility of analyzing the proposed approach with a broad range of workloads. The devised algorithm is meta algorithm, which can be utilized with any probable policy set.

Results/considerations The outcome of the experiments shows that the proposed algorithm performs better than the existing algorithms. The average regret of the algorithm is approximately 34 times superior in comparison with average regret across policies. The extension of the proposed model can be used for solving other different types of resource allocation problems like renting small vs. medium vs. large instances, selecting area of computing and different bounding alternatives in terms of memory, network, storage and CPU.

A generalization of the model, in which jobs have both length and size, and in which there are multiple units of a reusable goods available in each period, has been demonstrated in Ng et al. [37].

Approach The theory of mechanism design has been applied on combinatorial auction. Novelty of mechanisms lies in the fact that it permits matching of agent to a series of moves and the mechanism remains strategy proof.

Experimentation Sun's JXTA and Berkeley DB is used for implementation of the proposed mechanism Virtual Worlds and positive outcomes has been obtained. The pool is deployed on various servers to obtain scalability. The performance is validated with several different scenarios of resource allocation.

Results/considerations VIRTUAL WORLDS mechanism generalizes the online auction for the unit-length model. But, it was presented without competitive analysis and for a model in which agents cannot misreport their arrival time or patience.

Authors studied the problem of online scheduling of a single reusable resource over a finite-time period [40]. They characterize truthful allocation rules and derive lower-bound competitive ratios. Authors considered single-valued domains, and this approach is no longer incentive compatible for the setting where agents' preferences are described by a vector of values.

Approach The non-strategic setting has been used by authors. The assumption of the algorithm is that whatever the true characteristics of every job at its release are known by the algorithm. A distinct interested agent is the owner of each job. Each job is released to owning agent not to algorithm. Manipulation of the algorithm is done by agent in four different ways. It can decide the time of submission of the job to the algorithm; it can increase the length of the job arbitrarily. An arbitrary value and deadline for the job can be declared by the agent.

Experimentation In the experimental setting, a task from each agent is taken as input by the mechanism and a schedule for the job is returned as output. Every agent made payment to the center. The requirement of the design of mechanism to be incentive compatible, agent always submit its job immediately upon release with its best interest and correct declaration of its value, deadline and length is done by the agent.

Results/considerations Every incentive issue is addressed by the proposed mechanism with the increase in ratio of competition by one. A matching lower bound over the competitive ratio can be obtained through a deterministic mechanism in which agent never paid.

A combinatorial greedy allocation mechanism to find distribution of resources in cloud on the basis of sorted order of candidate for allocation has been proposed by authors [10]. Pricing mechanisms have also been designed for the same.

Approach These pricing mechanisms provide prices of buyer depending on the critical value, and the payments of seller which are computed through surplus distribution. This surplus distribution depends on the rules of payment for proportional value and direct value. Resource allocation problem of double-sided virtual resources is formulated through mathematical notations and integer program. The relationship of this formulation with the standard problems of optimization is provided to find the complexity class of the same.

Experimentation To analyze the quality of allocation by the devised greedy allocation scheme and to investigate seller strategic manipulation opportunity, several experi-

ments have been done for the simulation. The behavior of scheme is analyzed in a vast range of market scenario like variable number of sellers, several number of buyers created depending on the granularity of buyers, varying levels of resource provisioning and changing comparative ratio of price. Uniform distribution is used to create the price value of buyers and price reservation of sellers. Two hundred iterations are done for every scenario.

Results/considerations Depending on the experimental outcomes, authors claim that the devised mechanism satisfies budget balance, computational tractability, individual rationality, and truthfulness which are obtained for the buyers of common minded. With the proposed mechanism, computational complexity is reduced in comparison with other existing approaches. The mechanism is theoretically robust in nature and it works in a real setting of bi-sided cloud market.

An online lightweight auction mechanism has been proposed by authors for resource pricing and allocation dynamically in the real-world application of cloud [42].

Approach A game of sealed winner-bid auction has been proposed for allocation of cloud resources. Depending on their anticipated value upon scheduling, the game can specify bid price of users through a bid function which is based on valuation. Incoming time of tasks is heterogeneous, and in the scheduling process, scheduler is called at the occurrence of event of resource release. Due to this, it is called an online auction. Users are pushed to provide their actual bid at any time of scheduling due to design of method.

Experimentation Actual workload data are used in the experimentation, consisting of 20 logs taken from the archive of parallel workloads as well as grid workloads for getting reliable outcomes. The CloudSim simulator is employed for implementation of the experiment [9]. The configuration of the PC used in the simulation is 8 GB RAM, 1 TB HDD and Intel 3.00 GHz Dual Core CPU. Experimental outcomes are validated for every log of 20 workloads with user budget of five distinct combinations. Hence, for every algorithm total 100 simulations are done.

Results/considerations With the proposed algorithms, efficiency of cloud resources is increased. The number of executed job is also increased, and the profit of both cloud providers as well as users is enhanced. After comparing the devised mechanism with six other existing mechanisms with respect to payable price, number of executed tasks, profit of provider, use of resources and makespan, it is found that the performance of the devised mechanism is better.

To efficiently schedule bag of tasks (BoT) (independent tasks of small, medium and large scale), a bio-inspired algorithm has been devised by authors [15].

Approach Gaussian distribution is used to model the average execution time of BoT tasks on VMs through the mechanism of modified particle swarm optimization (MPSO). The concept of artificial neural network (ANN) is used to forecast the spot instances' future values by using the past data of a particular region. The validation of these values is done with real spot values. Both types of instances with changeable price structure, such as on-demand and spot, are used in the article.

Experimentation CloudAnalyst simulator and Python-based simulator are used for the experimentation of the proposed algorithm. IaaS-based cloud computing environment

is simulated by CloudAnalyst simulator. Real-time scenes of six different geographical locations are provided by CloudAnalyst simulator. Through a particular application, users from a specific geographical location can be recognized. Python-based simulator is used for the experimentation of devised bio-inspired MPSO algorithm. Four interconnected physical machines of different configurations are used in the experimental setup. In these four machines, two machines work as a server, one machine works as a task sender and one machine behaves like a load balancer with various VMs that runs the incoming tasks. Tasks needed several resources like memory and CPU. These resources are provided by the pool of cloud resources.

Results/considerations The devised MPSO algorithm performs better by checkpointing the VM's state in comparison with other existing algorithms. The parameters taken into consideration for measuring the performance are average response time, task execution time and maintaining the equal load among the VM instances. Employing spot instances reduces cost 38.23% in comparison with on-demand instances.

3.4.1 Conclusion of the literature review on enhancing reliability

Reliability can be enhanced by analyzing spot market bidding policies, virtualization mechanism, understanding price dynamics, optimizing cost and through online auction frameworks. SipaaS tool is devised by authors for the implementation of price dynamics to offer web services for price auction and spot instance reservation depending on the set of RESTful services. Several bidding mechanisms like AMAZING, ABMP, Q-learning based have been proposed by authors for enhancing the reliability of job execution on spot instances.

4 State-of-the-art challenges and research directions

The major problem in spot instances is how to delay spot instance revocation as much as possible. It can be done by exact forecasting of spot price so that user can bid accordingly and how to mitigate the effect of spot instance revocation by different fault-tolerant techniques like checkpointing, migration and task duplication.

Hence, the two research directions in the field of preemptible VMs in cloud computing are to increase reliability and fault tolerance in the use of spot instances. For increasing reliability, one has to develop an efficient technique of spot price prediction to facilitate the users in bidding. To increase fault-tolerant level, one can implement a fault-tolerant architectures like MapReduce, grid, queue-based and checkpointing.

Checkpointing is preferred because it does not require major changes to client applications and checkpointing can also be merged with other fault-tolerant techniques for making a reliable system. MapReduce is not suitable for real-time and iterative data processing. Grid can be complex and costly fault-tolerant technique. There is requirement of more time in queue-based technique in comparison with others. Checkpointing mechanism results in extra overhead which in turn increase the job execution time. This overhead must be minimized to reduce time and cost of task execution through spot instances.

Several strategies have been proposed by various authors for increasing reliability and fault-tolerant level of task execution through preemptible VMs. These strategies are discussed in Sect. 3 in detail. There is always scope of some improvement in these already proposed mechanisms to further enhance the level of fault tolerance and reliability in the use of preemptible VMs.

5 Conclusions and future work

In this article, we have analyzed the research directions in the use of preemptible VMs in cloud computing. First introduction is given by incorporating motivation, focus and goals. Then some basic concepts needed to understand the area are presented. Further, the state-of-the-art survey of preemptible VMs and their pricing are given. Though the concept of preemptible VMs provides less cost resources, still there are some problem in them. We found two research directions in the area of spot instances; these are increasing reliability and improving fault-tolerant level in the use of preemptible VMs. Reliability can be enhanced by accurate prediction of bid price. Several techniques are discussed in Sect. 3 for enhancing the reliability of preemptible VMs. These techniques can be further optimized to increase the reliability level of spot instances. Fault tolerance can be improved by implementation of some fault-tolerant architecture like MapReduce, grid, queue-based and checkpointing. Checkpointing is preferred among other fault-tolerant techniques because of its flexible nature. The overhead involved in checkpointing technique must be minimum.

In the future, our attempt will be toward development of various algorithms. One of them will be for price prediction of preemptible VMs. A bidding strategy will be proposed in another algorithm. Further attempts will be made toward the development of a checkpointing algorithm which will reduce the number of checkpoints taken during the tenure of job execution through spot instances. The performance of the proposed algorithms will be compared to show that the proposed algorithms are better in comparison with other techniques discussed in Sect. 3.

References

1. (2017) Cloud computing. https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/. Accessed 25 Dec 2017
2. (2018) Amazon EC2 instance types. https://aws.amazon.com/ec2/instance-types/. Accessed 25 Mar 2018
3. Abundo M, Valerio VD, Cardellini V, Presti FL (2015) QoS-aware bidding strategies for VM spot instances: a reinforcement learning approach applied to periodic long running jobs. In: IFIP/IEEE International Symposium on Integrated Network Management, IM 2015, Ottawa, ON, Canada, 11–15 May, 2015, pp 53–61. https://doi.org/10.1109/INM.2015.7140276
4. Agmon Ben-Yehuda O, Ben-Yehuda M, Schuster A, Tsafrir D (2013) Deconstructing amazon EC2 spot instance pricing. ACM Trans Econ Comput 1(3):16. https://doi.org/10.1145/2509413.2509416
5. Ben-Yehuda OA, Ben-Yehuda M, Schuster A, Tsafrir D (2011) Deconstructing amazon EC2 spot instance pricing. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, pp 304–311. https://doi.org/10.1109/CloudCom.2011.48
6. Blum A, Sandholm T, Zinkevich M (2006) Online algorithms for market clearing. J ACM 53(5):845–879. https://doi.org/10.1145/1183907.1183913

7. Burgess M, Wiedenbeck B (2010) Strategic bidding on amazon EC2. https://scholar.google.co.in/scholar?hl=en&as_sdt=0,5&cluster=3572391033188216107
8. Buyya R, Vecchiola C, Selvi ST (2013) Mastering cloud computing: foundations and applications programming. Newnes. https://doi.org/10.1016/C2012-0-06719-1
9. Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exper 41(1):23–50. https://doi.org/10.1002/spe.995
10. Chichin S, Vo QB, Kowalczyk R (2017) Towards efficient and truthful market mechanisms for double-sided cloud markets. IEEE Trans Serv Comput 10(1):37–51
11. Chohan N, Castillo C, Spreitzer M, Steinder M, Tantawi AN, Krintz C (2010) See spot run: using spot instances for mapreduce workflows. In: 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010. https://www.usenix.org/conference/hotcloud-10/see-spot-run-using-spot-instances-mapreduce-workflows
12. Dawoud W, Takouna I, Meinel C (2012) Increasing spot instances reliability using dynamic scalability. In: 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24–29, 2012, pp 959–961. https://doi.org/10.1109/CLOUD.2012.58
13. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, subramanian SS, Vosshall P, Vogels W (2007) Dynamo: amazon's highly available key-value store. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14–17, 2007, pp 205–220. https://doi.org/10.1145/1294261.1294281
14. Di S, Robert Y, Vivien F, Kondo D, Wang C, Cappello F (2013) Optimization of cloud task processing with checkpoint-restart mechanism. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA—November 17–21, 2013, pp 64:1–64:12. https://doi.org/10.1145/2503210.2503217
15. Domanal SG, Reddy GRM (2018) An efficient cost optimized scheduling for spot instances in heterogeneous cloud environment. Future Gener Comput Syst 84:11–21
16. Friedman EJ, Parkes DC (2003) Pricing wifi at starbucks: issues in online mechanism design. In: Proceedings of the 4th ACM Conference on Electronic Commerce, ACM, New York, NY, USA, EC '03, pp 240–241. https://doi.org/10.1145/779928.779978
17. Gong Y, He B, Zhou AC (2015) Monetary cost optimizations for MPI-based HPC applications on amazon clouds: checkpoints and replicated execution. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15–20, 2015, pp 32:1–32:12. https://doi.org/10.1145/2807591.2807612
18. Guo W, Chen K, Wu Y, Zheng W (2015) Bidding for highly available services with low price in spot instance market. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, NY, USA, HPDC '15, pp 191–202. https://doi.org/10.1145/2749246.2749259
19. He X, Shenoy P, Sitaraman R, Irwin D (2015) Cutting the cost of hosting online services using cloud spot markets. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, NY, USA, HPDC '15, pp 207–218. https://doi.org/10.1145/2749246.2749275
20. Huang B, Jarrett NWD, Babu S, Mukherjee S, Yang J (2015) Cumulon: matrix-based data analytics in the cloud with spot instances. PVLDB 9(3):156–167. http://www.vldb.org/pvldb/vol9/p156-huang.pdf
21. Jangjaimon I, Tzeng N (2013) Adaptive incremental checkpointing via delta compression for networked multicore systems. In: 27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20–24, 2013, pp 7–18. https://doi.org/10.1109/IPDPS.2013.33
22. Jangjaimon I, Tzeng N (2015) Effective cost reduction for elastic clouds under spot instance pricing through adaptive checkpointing. IEEE Trans Comput 64(2):396–409. https://doi.org/10.1109/TC.2013.225
23. Javadi B, Thulasiram RK, Buyya R (2011) Statistical modeling of spot instance prices in public cloud environments. In: IEEE 4th International Conference on Utility and Cloud Computing, UCC 2011, Melbourne, Australia, December 5–8, 2011, pp 219–228. https://doi.org/10.1109/UCC.2011.37
24. Javadi B, Thulasiram RK, Buyya R (2013) Characterizing spot price dynamics in public cloud environments. Future Gener Comput Syst 29(4):988–999. https://doi.org/10.1016/j.future.2012.06.012

25. Jia Q, Shen Z, Song W, van Renesse R, Weatherspoon H (2016) Smart spot instances for the supercloud. In: Proceedings of the 3rd Workshop on CrossCloud Infrastructures and Platforms, ACM, New York, NY, USA, CrossCloud '16, pp 5:1–5:6. https://doi.org/10.1145/2904111.2904114
26. Jung D, Lim J, Yu H, Suh T (2014) Estimated interval-based checkpointing (EIC) on spot instances in cloud computing. J Appl Math 2014:217,547:1–217,547:12. https://doi.org/10.1155/2014/217547
27. Kaminski B, Szufel P (2015) On optimization of simulation execution on amazon EC2 spot market. Simul Model Pract Theory 58:172–187. https://doi.org/10.1016/j.simpat.2015.05.008
28. Kaminski B, Szufel P (2015) On optimization of simulation execution on amazon EC2 spot market. Simul Modell Pract Theory 58(Part 2):172–187. doi: 10.1016/j.simpat.2015.05.008
29. Karunakaran S, Krishnaswamy V, Sundarraj RP (2014) Decisions models and opportunities in cloud computing economics: a review of research on pricing and markets. Springer, Cham, pp 85–99. https://doi.org/10.1007/978-3-319-07950-9_7
30. Khatua S, Mukherjee N (2013a) Application-centric resource provisioning for amazon EC2 spot instances. Springer, Berlin, pp 267–278. https://doi.org/10.1007/978-3-642-40047-6_29
31. Khatua S, Mukherjee N (2013b) A novel checkpointing scheme for amazon EC2 spot instances. In: 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013, Delft, Netherlands, May 13–16, 2013, pp 180–181. https://doi.org/10.1109/CCGrid.2013.71
32. Marathe A, Harris R, Lowenthal DK, de Supinski BR, Rountree B, Schulz M (2014) Exploiting redundancy for cost-effective, time-constrained execution of HPC applications on amazon EC2. In: The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada—June 23–27, 2014, pp 279–290. https://doi.org/10.1145/2600212.2600226
33. Mattess M, Vecchiola C, Buyya R (2010) Managing peak loads by leasing cloud infrastructure services from a spot market. In: 2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC), pp 180–188. https://doi.org/10.1109/HPCC.2010.77
34. Mazzucco M, Dumas M (2011) Achieving performance and availability guarantees with spot instances. In: 2011 IEEE International Conference on High Performance Computing and Communications, pp 296–303. https://doi.org/10.1109/HPCC.2011.46
35. Mell P, Grance T (2011) The NIST definition of cloud computing. Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
36. Menache I, Shamir O, Jain N (2014) On-demand, spot, or both: dynamic resource allocation for executing batch jobs in the cloud. In: 11th International Conference on Autonomic Computing (ICAC 14), USENIX Association, Philadelphia, PA, pp 177–187. https://www.usenix.org/conference/icac14/technical-sessions/presentation/menache
37. Ng C, Parkes DC, Seltzer M (2003) Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. In: Proceedings of the 4th ACM Conference on Electronic Commerce, ACM, New York, NY, USA, EC '03, pp 238–239. https://doi.org/10.1145/779928.779977
38. Poola D, Ramamohanarao K, Buyya R (2014) Fault-tolerant workflow scheduling using spot instances on clouds. In: Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10–12 June, 2014, pp 523–533. https://doi.org/10.1016/j.procs.2014.05.047
39. Poola D, Ramamohanarao K, Buyya R (2016) Enhancing reliability of workflow execution using task replication and spot instances. TAAS 10(4):30:1–30:21. https://doi.org/10.1145/2815624
40. Porter R (2004) Mechanism design for online real-time scheduling. In: Proceedings of the 5th ACM Conference on Electronic Commerce, ACM, New York, NY, USA, EC '04, pp 61–70. https://doi.org/10.1145/988772.988783
41. Qu C, Calheiros RN, Buyya R (2016) A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. J Netw Comput Appl 65:167–180. https://doi.org/10.1016/j.jnca.2016.03.001
42. Salehan A, Deldari H, Abrishami S (2017) An online valuation-based sealed winner-bid auction game for resource allocation and pricing in clouds. J Supercomput 73(11):4868–4905
43. Sharma P, Lee S, Guo T, Irwin D, Shenoy P (2015) Spotcheck: designing a derivative IaaS cloud on the spot market. In: Proceedings of the Tenth European Conference on Computer Systems, ACM, New York, NY, USA, EuroSys '15, pp 16:1–16:15. https://doi.org/10.1145/2741948.2741953
44. Sharma P, Irwin D, Shenoy P (2016) How not to bid the cloud. In: Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing, USENIX Association, Berkeley, CA, USA, HotCloud'16, pp 1–6. http://dl.acm.org/citation.cfm?id=3027041.3027042

45. Shroff G (2010) Enterprise cloud computing: technology, architecture, applications. Cambridge University Press, Cambridge
46. Singh VK, Dutta K (2015) Dynamic price prediction for amazon spot instances. In: 48th Hawaii International Conference on System Sciences, HICSS 2015, Kauai, Hawaii, USA, January 5–8, 2015, pp 1513–1520. https://doi.org/10.1109/HICSS.2015.184
47. Sinha PK (1996) Distributed operating systems: concepts and design, 1st edn. Wiley-IEEE Press, Hoboken
48. Sinha PK (2012) Distributed operating systems: concepts and design. PHI Learning Pvt, Ltd, New Delhi
49. Song Y, Zafer M, Lee KW (2012) Optimal bidding in spot instance market. In: 2012 Proceedings IEEE INFOCOM, pp 190–198. https://doi.org/10.1109/INFCOM.2012.6195567
50. Subramanya S, Guo T, Sharma P, Irwin D, Shenoy P (2015) Spoton: a batch computing service for the spot market. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, ACM, New York, NY, USA, SoCC '15, pp 329–341. https://doi.org/10.1145/2806777.2806851
51. Taifi M (2011) ACM SRC poster: Spotmpi: auction-based high performance cloud computing. In: Conference on High Performance Computing Networking, Storage and Analysis—Companion Volume, SC 2011, Seattle, WA, USA, November 12–18, 2011, pp 115–116. https://doi.org/10.1145/2148600.2148660
52. Taifi M, Shi JY, Khreishah A (2011) Spotmpi: A framework for auction-based HPC computing using amazon spot instances. In: Algorithms and Architectures for Parallel Processing—11th International Conference, ICA3PP, Melbourne, Australia, October 24–26, 2011, proceedings, part II, pp 109–120. https://doi.org/10.1007/978-3-642-24669-2_11
53. Tang S, Yuan J, Li Xy (2012a) Amazing: An optimal bidding strategy for amazon EC2 cloud spot instance. https://scholar.google.co.in/scholar?hl=en&as_sdt=0%2C5&q=AMAZING%3A+An+Optimal+Bidding+Strategy+for+Amazon+EC2+cloud+spot+instances&btnG=
54. Tang S, Yuan J, Li XY (2012b) Towards optimal bidding strategy for amazon EC2 cloud spot instance. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp 91–98. https://doi.org/10.1109/CLOUD.2012.134
55. Tang S, Yuan J, Wang C, Li X (2014) A framework for amazon EC2 bidding strategy under SLA constraints. IEEE Trans Parallel Distrib Syst 25(1):2–11. https://doi.org/10.1109/TPDS.2013.15
56. Toosi AN, Khodadadi F, Buyya R (2016) SipaaS: spot instance pricing as a service framework and its implementation in openstack. Concurr Comput Pract Exp 28(13):3672–3690. https://doi.org/10.1002/cpe.3749
57. Veena K, Anand C, Gupta CP (2016) Temporal and spatial trend analysis of cloud spot instance pricing in amazon EC2. In: 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, 14th International Conference on Pervasive Intelligence and Computing, 2nd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2016, Auckland, New Zealand, August 8–12, 2016, pp 909–912. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.157
58. Voorsluys W, Buyya R (2012) Reliable provisioning of spot instances for compute-intensive applications. In: IEEE 26th International Conference on Advanced Information Networking and Applications, AINA, 2012, Fukuoka, Japan, March 26–29, 2012, pp 542–549. https://doi.org/10.1109/AINA.2012.106
59. Wallace RM, Turchenko V, Sheikhalishahi M, Turchenko I, Shults V, Vazquez-Poletti JL, Grandinetti L (2013) Applications of neural-based spot market prediction for cloud computing. In: 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), vol 02, pp 710–716. https://doi.org/10.1109/IDAACS.2013.6663017
60. Wang P, Qi Y, Hui D, Rao L, Liu X (2013) Present or future: optimal pricing for spot instances. In: 2013 IEEE 33rd International Conference on Distributed Computing Systems, pp 410–419. https://doi.org/10.1109/ICDCS.2013.68
61. Xu H, Li B (2013) A study of pricing for cloud resources. SIGMETRICS Perform Eval Rev 40(4):3–12. https://doi.org/10.1145/2479942.2479944
62. Xu Z, Stewart C, Deng N, Wang X (2016) Blending on-demand and spot instances to lower costs for in-memory storage. In: 35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10–14, 2016, pp 1–9. https://doi.org/10.1109/INFOCOM.2016.7524348

63. Yi S, Kondo D, Andrzejak A (2010) Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In: IEEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, 5–10 July, 2010, pp 236–243. https://doi.org/10.1109/CLOUD.2010.35
64. Yi S, Andrzejak A, Kondo D (2012) Monetary cost-aware checkpointing and migration on amazon cloud spot instances. IEEE Trans Serv Comput 5(4):512–524. https://doi.org/10.1109/TSC.2011.44
65. Zafer M, Song Y, Lee KW (2012) Optimal bids for spot VMS in a cloud for deadline constrained jobs. In: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, IEEE Computer Society, Washington, DC, USA, CLOUD '12, pp 75–82. https://doi.org/10.1109/CLOUD.2012.9
66. Zaman S, Grosu D (2011) Efficient bidding for virtual machine instances in clouds. In: 2011 IEEE 4th International Conference on Cloud Computing, pp 41–48. https://doi.org/10.1109/CLOUD.2011.49
67. Zhao H, Pan M, Liu X, Li X, Fang Y (2012) Optimal resource rental planning for elastic applications in cloud market. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IEEE Computer Society, Washington, DC, USA, IPDPS '12, pp 808–819. https://doi.org/10.1109/IPDPS.2012.77
68. Zheng L, Joe-Wong C, Tan CW, Chiang M, Wang X (2015) How to bid the cloud. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, ACM, New York, NY, USA, SIGCOMM '15, pp 71–84. https://doi.org/10.1145/2785956.2787473