J Grid Computing (2019) 17:385–417
https://doi.org/10.1007/s10723-017-9424-0

Resource Provisioning Based Scheduling Framework for Execution of Heterogeneous and Clustered Workloads in Clouds: from Fundamental to Autonomic Offering

Sukhpal Singh Gill · Rajkumar Buyya

Received: 1 August 2017 / Accepted: 20 December 2017 / Published online: 19 January 2018
© Springer Science+Business Media B.V., part of Springer Nature 2018

Abstract Provisioning of adequate resources to cloud workloads depends on the Quality of Service (QoS) requirements of these cloud workloads. Based on workload requirements (QoS) of cloud users, discovery and allocation of best workload-resource pair is an optimization problem. Acceptable QoS can be offered only if provisioning of resources is appropriately controlled. So, there is a need for a QoS-based resource provisioning framework for the autonomic scheduling of resources to observe the behavior of the services and adjust it dynamically in order to satisfy the QoS requirements. In this paper, framework for self-management of cloud resources for execution of clustered workloads named as SCOOTER is proposed that efficiently schedules the provisioned cloud resources and maintains the Service Level Agreement (SLA) by considering properties of self-management and the maximum possible QoS parameters are required to improve cloud based services. Finally, the performance of SCOOTER has been evaluated in a cloud environment that demonstrates the optimized QoS parameters such as execution cost, energy consumption, execution time, SLA violation rate, fault detection rate, intrusion detection rate, resource utilization, resource contention, throughput and waiting time.

Keywords Cloud computing · Cloud workloads · Resource provisioning · Resource scheduling · Quality of service · Autonomic computing · Service level agreement · Self-management · Self-healing · Self-configuring · Self-optimizing · Self-protecting · Quality of service · Resource management · Autonomic cloud · E-commerce as a cloud service

1 Introduction

Cloud computing is a new paradigm that provides on-demand services over the Internet. Cloud services are viewed as a composition of distributed components and are offered as: Infrastructure (hardware, storage, and network), Platform or Software [1]. As cloud offers these three types of services, it requires Quality of Service (QoS) to efficiently monitor and measure the delivered services and further needs to follow Service Level Agreements (SLAs). However, providing dedicated cloud services that ensure user's dynamic QoS requirements and avoid SLA violations is a big challenge in cloud computing [2]. Currently, cloud services are provisioned and scheduled according to resources' availability without ensuring the expected performances [3]. The cloud provider should evolve its ecosystem in order to meet QoS requirements of

S. S. Gill (✉) · R. Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
e-mail: sukhpal.gill@unimelb.edu.au

R. Buyya
e-mail: rbuyya@unimelb.edu.au

each cloud component. To realize this, there is a need to consider two important aspects which reflect the complexity introduced by the cloud management: QoS-aware and autonomic management of cloud services [4]. QoS-aware aspect expects a service to be aware of its behavior to ensure the high availability, reliability of service, minimum execution cost, minimum execution time, maximum energy efficiency and so forth. Autonomic management implies the fact that the service is able to self-manage itself as per its environment needs. Thus, maximizing cost-effectiveness and utilization for applications while ensuring performance and other QoS guarantees, requires leveraging important and extremely challenging tradeoffs [5]. Based on human guidance, an autonomic system keeps the system stable in unpredictable conditions and adapts quickly in new environmental conditions like software, hardware failures and so forth. Intelligent autonomic systems work on the basis of QoS parameters and are inspired by biological systems that can easily handle the problems like uncertainty, heterogeneity and dynamism. Autonomic cloud computing system can be autonomic model that considers four steps of autonomic system (Monitor, Analyze, Plan and Execute) in a control loop and the goal of intelligent autonomic system is to execute application within deadline by fulfilling QoS requirements as described by user with minimum complexity. Based on QoS requirements, an intelligent autonomic system provides self-management of resources which considers following properties [1–5] of self-management: a) *self-healing* is a capability of an autonomic system to identify, analyze and recover from unfortunate faults automatically, b) *self-configuring* is a capability of an autonomic system to adapt to the changes in the environment, c) *self-optimizing* is a capability of an autonomic system to improve the performance and d) *self-protecting* is a capability of an autonomic system to protect against intrusions and threats.

1.1 Previous Contributions

In earlier work [6–18], different cloud workloads have been identified and categorized based on their QoS characteristics and QoS metrics have been identified for each workload and have been analyzed for creating better application design [6]. Next, various research issues related to QoS and SLA for management of cloud resources for workload execution have been identified [7]. To develop a resource provisioning technique, methodical survey [8] has been done to evaluate and uncover the research challenges based on available existing research in the field of cloud resource provisioning and based on these challenges, a QoS based resource provisioning technique (Q-aware) has been developed to map the resources to the workloads based on user requirements [9]. The main aim of Q-aware is to analyze the workloads, categorize them on the basis of common patterns and then provision the resources for execution of cloud workloads before actual resource scheduling. The simulation based experimental results demonstrate that Q-aware is efficient in reducing execution time and execution cost of cloud workloads along with other QoS parameters.

Further, the resource scheduling algorithms have been explored in detail [10] through methodical survey and compared the resource scheduling algorithms based on important aspects of resource scheduling to identify research issues and open challenges still unresolved. Based on research issues and open challenges, resource scheduling framework (QRSF) has been proposed, in which resources have been scheduled by using different resource scheduling policies (cost, time, cost-time and bargaining based) [11]. The simulation based experimental results show that the QRSF gives better results in terms of energy consumption, execution cost and execution time of different cloud workloads as compared to existing scheduling frameworks. In QRSF, there was direct scheduling of resources which further needs lot of work every time to schedule resources to execute workloads by fulfilling their QoS requirements due to the absence of QoS-based resource provisioning before actual scheduling of resources. To solve that problem, the concept of QRSF has been further extended by presenting QoS based Resource Provisioning and Scheduling (QRPS) Framework [12]. In QRPS framework, QoS based resource provisioning technique (Q-aware) is used for resource provisioning in which: i) clustering of workloads is done using k-means clustering algorithm after assigning weights to quality attributes of each workload and ii) resources are provisioned based on their QoS requirements. Further, four different resource scheduling policies are used to schedule the provisioned resources. Further, the performance of QRPS has been tested using QoS parameters such as execution time, execution cost and energy consumption. Moreover, Q-aware has been extended by proposing BULLET i.e. Particle Swarm Optimization (PSO) based resource scheduling algorithm [19], which execute cloud workloads and improves the execution cost and time as compared to QRSF.

In QRPS, there was manual provisioning based resource scheduling, which further needs lot of human work every time to schedule resources to execute workloads by fulfilling their QoS requirements. To realize this, there is a need to consider an important aspect that reflects the complexity introduced by the cloud management: QoS-aware autonomic management of cloud services [13, 16]. To design a QoS based autonomic resource management framework, which can manage resources automatically and provides reliable, secure and cost-efficient cloud services, a comprehensive investigation has been conducted [14] to study various existing autonomic resource management techniques and properties of self-management (self-healing, self-configuring, self-optimizing, and self-protecting). Based on existing research challenges in the area of autonomic resource management, QRPS has been further extended by proposing energy-aware autonomic resource scheduling technique (EARTH), in which IBM's autonomic computing concept [21] has been used to schedule the resources automatically by optimizing energy consumption and resource utilization, where user can easily interact with the system using available user interface [15]. But, EARTH can execute only homogenous cloud workloads and the complexity of resource scheduling in EARTH increases with the increase of number of workloads. To address this issue, SOCCER [20] is proposed which clusters the heterogeneous cloud workloads and executes them with minimum energy consumption, but SCOEER only focuses on one aspect of self-optimization i.e. energy consumption. Further, SLA-aware autonomic resource management (STAR) technique [17] is proposed, which mainly focuses on other important aspect of self-optimization i.e. SLA violation rate and also analyzed the impact of QoS parameters on SLA violation rate. Next, QoS-aware autonomic resource management approach (CHOPPER) [18] has been proposed, which offers self-configuration of applications and resources, self-healing by handling sudden failures, self-protection against security attacks and self-optimization for maximum resource utilization.

In this research work, CHOPPER is extended by proposing a resource management framework called ***SCOOTER*** (**S**elf-management of **C**l**O**ud res**O**urces for execu**T**ion of clust**E**red wo**R**kloads) that efficiently schedules the provisioned cloud resources and maintains the SLA by considering four properties of self-management and the maximum possible QoS parameters (execution cost, energy, execution time, SLA violation rate, fault detection rate, intrusion detection rate, resource utilization, resource contention, throughput and waiting time) required to improve cloud based services. Finally, the proposed framework has been verified with the help of a case study of *e-commerce* in a cloud environment that demonstrates the optimized QoS parameters. SCOOTER improves user satisfaction and, increases reliability and availability of services. The rest of the paper is structured as follows: Section 2 presents related work of cloud resource management. Proposed framework is presented in Section 3. Section 4 describes the experimental setup used for performance evaluation and experimental results. Section 5 presents conclusions and future work.

2 Related Work

Provisioning and scheduling of resources in cloud has been done through different techniques existing in literature but QoS parameters are important factor that it is challenging to optimize automatically. To optimize QoS parameters, researchers proposed different techniques. *Resource provisioning* is defined as identifying adequate resources for a given workload based on QoS requirements described by cloud consumer [8] whereas *resource scheduling* is mapping and execution of cloud consumer workloads on provisioned resources through resource provisioning [10]. The objective of resource scheduling is to schedule the provisioned resources to the suitable workloads on time, so that applications can utilize the resources effectively [11]. In other words, the amount of resources should be minimum for a workload to maintain a desirable level of service quality, or maximize throughput (or minimize workload completion time) of a workload. The goal of autonomic resource provisioning and scheduling is to execute application within their deadline by fulfilling QoS requirements as described by user in SLA. Autonomic computing systems have capability to deal with all the situations dynamically and manage the situation in unpredictable environment [14]. Autonomic Cloud Computing Systems (ACCSs) control the working of cloud based systems and applications without human intervention. ACCSs check, monitor and respond according to the situation like self-healing, self-protecting and self-configuring and self-optimizing. This section is presenting the related work of resource provisioning and scheduling along with self-management in cloud computing.

2.1 QoS based Resource Provisioning

Andres et al. [22] proposed a Distributed Resource Management Framework (DRMF) to determine resource provisioning of workload on innovativeness clouds. To handle with erroneous request for resource that causes to over utilization of resources delivered by cloud workload request, their framework has revealed a design-based technique for approximating the workload execution time specified it's scheduling but the behavior of a particular workload was not identified. Christian et al. [23] proposed a Deadline based Resource Provisioning (DRP) technique using dynamic reallocation for execution of scientific workloads through Aneka in hybrid cloud environment. This technique decreases the makespan of workloads only without considering execution cost of resources. Nikolas et al. [24] proposed a Self-Adaptive Resource Provisioning (SARP) framework to find the appropriate predicting ways for a given perspective through the use of decision tree. This framework optimizes QoS parameters like relative error and SLA violation but does not consider execution time and execution cost as a QoS parameter. Hamid et al. [25] proposed a Dynamic Behavior based Resource Provisioning (DBRP) framework to assign adequate number of resources to workloads. This framework reduces the execution cost of resources only without considering execution time.

2.2 QoS based Resource Scheduling

Varalakshmi et al. [26] proposed an Optimal Workflow based Scheduling (OWS) framework to discover a solution that tries to meet the user-desired QoS constraint i.e. execution time. This framework shows slight improvement in resource utilization is attained. But it did not consider cost and energy as QoS parameters. Li et al. [27] proposed an Ant Colony Optimization based Job Scheduling (ACOJS) framework, which adapts to dynamic characteristics of cloud computing and incorporates particular benefits of ACO in NP-hard problems. This framework reduced only job completion time based on pheromone. Topcuoglu et al. [28] presented the Heterogeneous Earliest Time First (HEFT) framework to discover the average execution time of each workload and also the average communication time among the resources of two workloads. Then workloads in the workflow are well-ordered based on a rank function. The workload with higher rank value is given higher priority. In the resource selection stage workloads are scheduled in priorities and each workload is allocated to the resource that complete the workload at the earliest time. But it did not consider cost and time as QoS parameters. Pandey et al. [29] introduced a Particle Swarm Optimization based Heuristic (PSOH) framework to schedule the applications to cloud resources that proceeds both computation and data transmission cost. It is used for workflow applications by changing its computation and communication costs. The assessment results show that PSOH can reduce the cost and good sharing of workload onto resources, but it did not consider execution time of workloads.

2.3 QoS Based Autonomic Resource Provisioning and Scheduling

Self-management in cloud computing has four properties: a) self-healing, b) self-configuring, c) self-optimizing and d) self-protecting. It is a challenge to implement all the properties of self-management together and based on requirements and goals of an autonomic system, mostly some of the properties are considered.

2.3.1 Self-healing

The aim of *self-healing* is to make the necessary changes to recover from the faults occurred to maintain the working of system without any disruption. System must ensure that the successful execution of workloads or application without affecting its performance even in case of software, network and hardware faults. Software fault may occur due to unhandled exception occurs in high resource intensive workloads; other reasons may be deadlock, lesser storage space, unavailability of resources etc. Hardware fault may occur due to problem in hardware components like processor, RAM, HDD etc. Network faults may occur due to lack of scalability, physical damage, network breakage in case of distributed networks. Application Service Provider (ASP) [30] uses WSDL (Web Service Description Language) and Web Interface (HTTP) to design proactive and reactive heuristic policies to get an optimal solution. All the important QoS parameters are mentioned in SLA document. In this autonomic system, performance history is used to resolve the alerts generated at runtime due to some QoS parameters. ASP provides the feature of load balancing and VM allocation at runtime through the use of fully controlled autonomic loop. In this system, lease cost and SLA violations are reduced but very large execution time. Linlin at al. [31] proposed SLA-based Resource Allocation (SRA) mechanism to map the user requests to available resource, fulfill QoS requirements of user application at runtime and implemented in virtual environment. Further, SRA considers QoS parameters like response time, service time, cost and SLA violations. But SRA failed to analyze the effect of QoS parameters on SLA violation rate.

2.3.2 Self-configuring

The main aim of *self-configuring* is installation of missed or outdated components based on the alert generated by system without human intervention. Some components may be reinstalled in changing conditions and other components need updates. Self-configuring is also taking care of cost, which includes cost of resource and penalty cost in case of SLA violation. Case Base Reasoning (CBR) based framework uses human based interaction to make an agreement between user and provider called SLA for successful execution of workloads by considering resource utilization and scalability as a QoS requirement. In this system, various elastic levels are defined, and a control loop is used to enable the autonomic computing in virtual environment. Knowledge base stores the rules used in decision making after monitoring data (real and synthetic workloads) for resource configuration. This system executes in four steps: a) retrieves the most similar case, b) solves the problem through similar case, c) revises the solution and d) stores the key-features of solution into knowledge database for future use. SLA violations and resource utilization are improved in this autonomic system without considering basic QoS parameters like cost, time, energy etc. Self-COnfigured, Cost-based Cloud qUery Services (COCCUS) [33] used centralized architecture to provide the query based facility, in which user can ask query regarding scheduling policies, priorities and budget information. CloudDBMS is used to store the information about the scheduling policies and user queries for further use. Main objectives of COCCUS are: i) to get and execute the user queries, ii) to store the queries in the data structure and iii) to minimize the maintenance cost of query execution but it has large execution time.

2.3.3 Self-optimizing

The main aim of *self-optimizing* is to map the tasks or workloads on appropriate resources using dynamic scheduling technique. Dynamic scheduling continually checks the status of execution and improves the system performance based on the feedback given by autonomic element. For data intensive applications, adaptive scheduling is used, which can be easily adapted in changing environment. Self-optimizing is affected by various QoS parameters such as execution time, execution cost, resource utilization, availability of service, reliability of service, energy efficiency and resource contention. Cloud Auto Scaling (CAS) [34] schedules activities of VM instance start up and shut-down automatically to improve the performance. CAS enables user to finish the execution of workloads or tasks within their deadline with minimum cost. Window Azure Platform is used to implement this autonomic system. CAS contains four components: history repository, performance monitor, VM manager and auto-scaling decider. Performance monitor checks the processing time, execution time and arrival time of workload. History repository is used to store the particular information about workload. VM manager maintains a relation between cloud providers and auto-scaling mechanism and executes plan of auto-scaling to finish the execution of workloads. CAS did not consider heterogeneous cloud workloads with different QoS parameters.

Autonomic Workload Manager (AWM) [35] used distributed provisioning and scaling decision making system to distribute the workloads on resources based on their common QoS characteristics. AWM divides resources into two categories: coarse-grained and fine-grained resources. AWM allocates the resources based on minimum response time and high throughput. This autonomic system executes in three steps: 1) allocate resources to workloads, 2) minimize execution time and 3) check execution status (if executes within time and budget then continues execution otherwise provide more resources). AWM is not able to determine the cost of execution of workloads. Mehdi et al. [36] proposed Autonomic Resource Contention Scheduling (ARCS) technique for distributed system to reduce resource contention in which more than one job shares same resource simultaneously. ARCS have four main components: i) front end policies (it performs admission control and queuing of jobs), ii) scheduler (it contains backfilling scheduling algorithm), iii) information service (information about scheduler) and (iv) back end policies (mapping of resources with jobs). ARCS established a relationship among layers of distributed resource management. ARCS did not check the variation of resource contention along with number of workloads. Energy-aware autonomic resource scheduling (EARTH) [15] is an autonomic resource management technique which schedules the resources automatically by optimizing energy consumption and resource utilization. Scheduling rules have been designed using the concept of fuzzy logic to calculate the priority of workload execution. Large number of rules is generated for every request, so it is very difficult to take an effective decision in timely manner. EARTH always executes the workloads with highest priority (which has earliest deadline), in which workloads with lowest priority is facing the problem of starvation.

2.3.4 Self-protecting

The main aim of *self-protecting* is to protect the system from malicious and intentional actions by tracking the doubtful activities and respond accordingly to maintain the working of system without any disruption. System should have knowledge about legal and illegal behavior to make distinction and apply operation accordingly to block the attack. Attack may be DDoS, R2L, U2R and Probing. In DDoS (Distributed Denial of Service) attack, huge traffic is generated by attackers to cause damage by flooding the victim's network. It includes SMURF (to create denial of service, attackers use Internet Control Message Protocol (ICMP) echo request by pointing packets toward broadcast IP address), LAND (Local Area Network Denial - when source and destination address is same, then attackers send spoofed SYN packet in TCP/IP network) and SYN Flood (attackers sending IP-spoofed packets which can crash memory). In Remote to Local (R2L), attackers access the system locally without authorization to damage the network by executing their commands. It includes attacks like IMAP (Internet Control Message Protocol), Password Guessing and SPY.

In User to Root (U2R), attackers get root access of the system to destroy the network. It includes attack like buffer overflow and rootkits. In Probing, attackers use programming language to steal the private information. It includes attacks like port sweep and NMAP (Network MAPper). Rainbow Architecture based Self-Protection (RASP) [37] is an autonomic technique in which security threats are detected at runtime through the use of patterns. RASP reduces the security breaching and improves the depth of defense. Detection rate of attacks in RASP is not as required. Self-Healing And self-Protection Environment (SHAPE) [38] is an autonomic system to recover from various faults (hardware, software, and network faults) and protect from security attacks (DDoS, R2L, U2R, and probing attacks). SHAPE is based on component based architecture, in which new components can be added or removed easily. Open source technologies are used to implement this autonomic system, but SHAPE is unable to execute heterogeneous workloads.

All the above research works have presented resource provisioning and scheduling in cloud computing without considering the important QoS parameters of self-management in a single autonomic resource management framework. In addition, SCOOTER needs to consider the basic features of cloud computing in order to execute the heterogeneous cloud workloads automatically optimize QoS parameters, which is not considered in other existing work.

3 SCOOTER: Self-Management of Cloud Resources for Execution of Clustered Workloads

Provisioning and scheduling of resources in cloud is an important part of resource management system. Mapping of cloud workloads to appropriate resources is mandatory to improve the performance of QoS parameters like execution time, execution cost and so forth. Based on QoS requirements, scheduling finds and maps the resources and workloads for execution. The existing scheduling frameworks execute the workloads without self-management. But in present scenario, there is a need of cloud based framework, which provisions and schedules computing resources automatically as well as adapt on the current situation dynamically by considering execution time, cost, energy, SLA violation rate, security, resource contention and so forth as a QoS parameters.

3.1 Problem Statement

A cloud provider needs automated and integrated intelligent strategies for provisioning of resources to offer services that are available, reliable and cost-efficient and thus achieve maximum resource utilization [1]. Resource provisioning in cloud is a complex task that is often compromised due to non-availability of the desired resources. The cloud services delivered by heterogeneous and dynamic nature of the cloud resources depend on the QoS [2]. Provisioning helps in identifying the kind and exact amount of resources. Once resources are provisioned, then scheduling can be done with the help of resource scheduling techniques. Literature reported that lot of work still needs to be done for optimal resource usage [3, 4]. Autonomic resource provisioning and scheduling technique can provide one of the solutions for optimal resource allocation by maximizing provider's revenues while satisfying customers QoS constraints, handle unexpected runtime situations automatically (e.g., unexpected delays in scheduling queues or unexpected failures) and thus minimizing resource usage cost and execution time. Maximizing the efficiency, dispersion, heterogeneity and uncertainty of resources brings challenges to resource allocation, which cannot be satisfied with traditional resource management techniques in cloud environment. To consider this problem, a set of self-regulating/independent cloud workloads $\{w_1, w_2, w_3, \ldots, w_x\}$ to map on a set of dynamic and heterogeneous resources $\{r_1, r_2, r_3, \ldots, r_n\}$ has been taken. R = $\{r_k,\ 1 \leq k \leq n\}$ is the collection of resources and n is the total number of resources. w = $\{w_i | 1 \leq i \leq x\}$ is the collection of cloud workloads and x is the total number of cloud workloads.

3.1.1 Objective and Commitments

The main objectives of this research work are:

a) To extend the existing research work and develop an autonomic resource provisioning framework for cloud resources based on user's QoS requirements.
b) To develop a resource scheduling approach that efficiently schedules the provisioned cloud resources and maintains the SLA.
c) To validate the developed framework in a cloud environment through a case study of E-commerce as a Cloud Service.

3.1.2 Objective Function

The goal of cloud provider is to minimize the SLA violation rate. The cloud workload will be executed only when the SLA violation rate is less than its threshold value (maximum SLA deviation agreed between cloud provider and user). *SLA Violation Rate* is the *product* of *Failure Rate* and *Weight of SLA* [17] and can be calculated as (1). List of SLA = $<m_1, m_2 \ldots\ldots\ldots\ldots\ldots.m_y>$, where y is total number of SLAs

$$Failure\,(m) = \begin{cases} m \ is \ not \ violtated, & Failure\,(m) = 1 \\ m \ is \ violated, & Failure\,(m) = 0 \end{cases}$$

$$Failure\ Rate = \sum_{i=1}^{y} \left(\frac{Failure\,(m_i)}{y} \right)$$

$$SLA\ Violation\ Rate = Failure\ Rate \times \sum_{i=1}^{y} (w_i) \tag{1}$$

Where w_i is weight for every SLA.

3.2 SCOOTER Architecture

SCOOTER efficiently schedules the provisioned cloud resources and maintains the SLA by considering four important properties of self-management and the maximum possible QoS parameters required to improve cloud based services. Architecture of SCOOTER is shown in Fig. 1, which comprises following units:

- *Cloud Workload Management Portal* (CWMP) acts as an interface for consumer to interact with

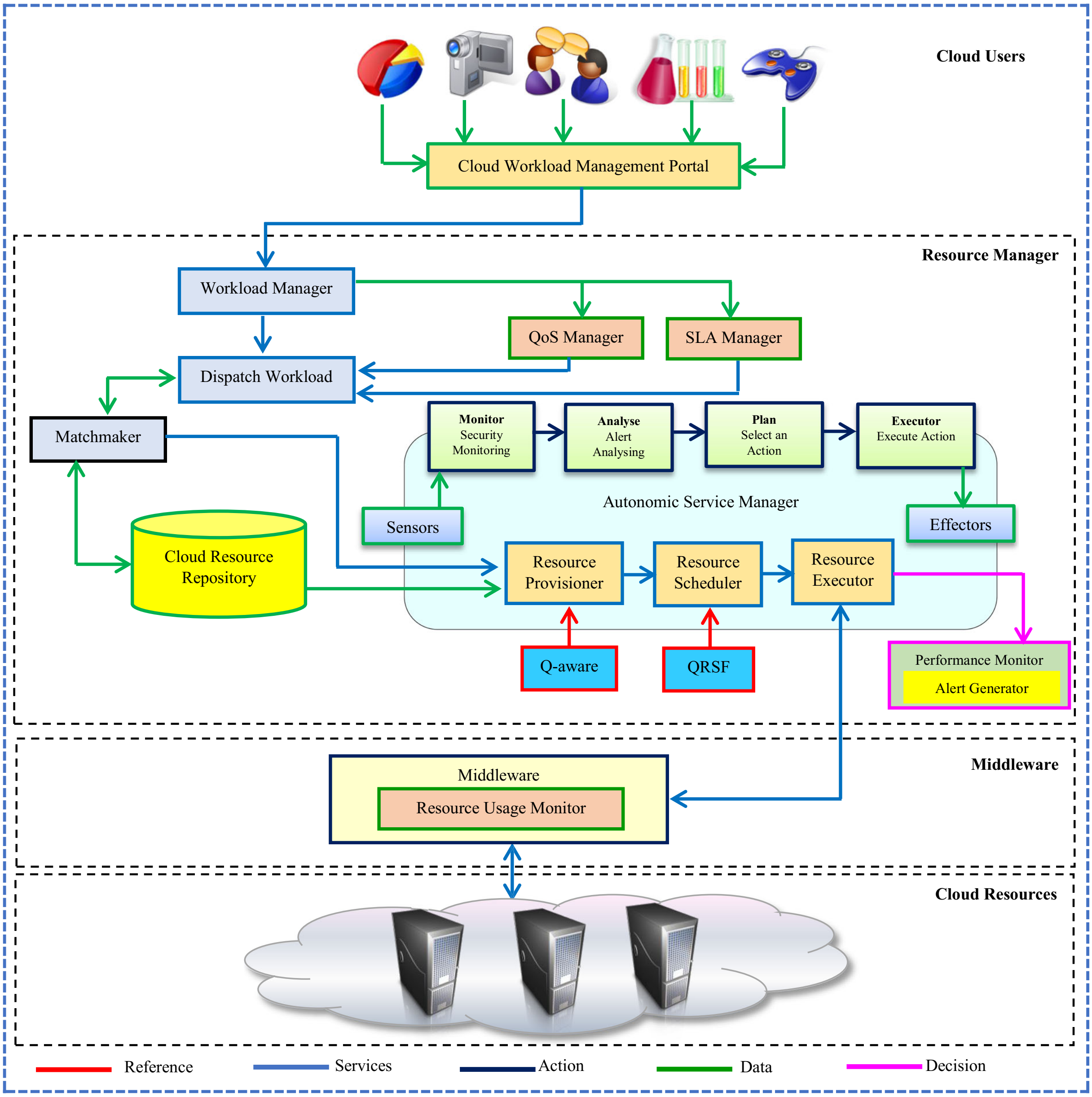


Fig. 1 SCOOTER architecture

SCOOTER for registration, login, ask queries and to submit workload details as shown in Figure 2. Based on the requirements of user, cloud provider generates schedule for workload execution.

- The aim of cloud *Workload Manager* is to look at different QoS requirements of cloud workloads to determine the feasibility of their execution. The different cloud workloads have different set of QoS requirements and characteristics. Workload manager also provides an input to SLA manager to find the required amount of cloud resources for workload execution. It also clusters the cloud workloads based on their QoS parameters.
- *QoS Manager* assesses the QoS requirements of the workloads. Based on the key QoS requirements of a specific workload, the QoS manager puts the workload into critical and non-critical queues through QoS assessment [18].

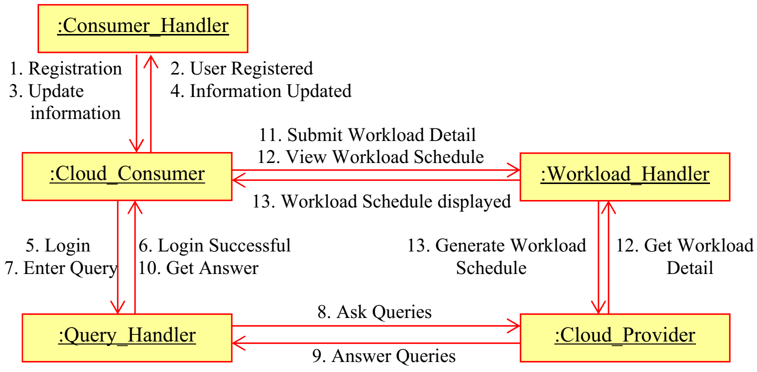


Fig. 2 Interaction of cloud user with CWMP

- *SLA Manager* prepares SLA document based on SLA information, which contains information about SLA violation (penalty rate and allowed maximum SLA deviation) and keep the record of urgent cloud workloads that would be placed in priority queue for earlier execution [17].
- *Matchmaker* identifies the required type of resources for provisioning of resources using the data available in cloud resource repository.
- *Cloud Resource Repository* contains the resource details including the number of CPUs used, size of memory, cost of resources, type of resources and number of resources. It also maintains the other description of resources (resource name, resource type, configuration, availability information, usage information and price of resource).
- *Autonomic Service Manager* comprises of six components of MAPE-K Loop: sensor, monitor, analyze, plan, executor and effector and the function of these components is described in Section 3.3.1.
- *Resource Provisioner* provisions the cloud resources, in which, suitable resources are identified for a specific workload based on their QoS requirements as designated by Matchmaker. These resources are then said to be provisioned as per the user requests (workloads).
- *Resource Scheduler* schedules the provisioned resources with maximum optimization of QoS parameters.
- *Resource Executor* executes the workloads using the scheduled resources.
- *Performance Monitor* measures the value of QoS parameters during workload execution and generate the alert in case of performance degradation.
- *Middleware* makes an interaction between resource manager and cloud infrastructure for execution of workloads. Resource Usage Monitor measures the value of resource utilization of different resources participating in execution of workloads.

3.3 Resource Provisioning and Scheduling

The process of resource provisioning and scheduling in cloud is shown in Fig. 3 and it contains following two phases of resource management:

a) Phase-1: Resource Provisioning

The objective of *Phase 1* is to provision the cloud resources using Q-aware [9] i.e. QoS based resource provisioning technique. Initially, cloud consumer submits the workload detail to workload analyzer for workload analysis along with workload information like QoS attributes, SLA and so forth. All the submitted workloads are considered as bulk of workloads. All the workloads should have their key QoS requirements, based on that, the workload is executed with some user defined constraints such as deadline, budget etc.

The types of workload that have been identified during workload analysis [6], are web apps, technical computing, endeavor software, performance testing, online transaction processing, e-commerce, central financial services, storage and backup services, production applications, software/project development and testing, graphics oriented, critical internet applications and mobile computing services. *Workload Analyzer* clusters the submitted cloud workloads based on workload patterns [6, 9] and Table 1 shows the outcome of pattern based clustering of workloads along with their QoS requitements.

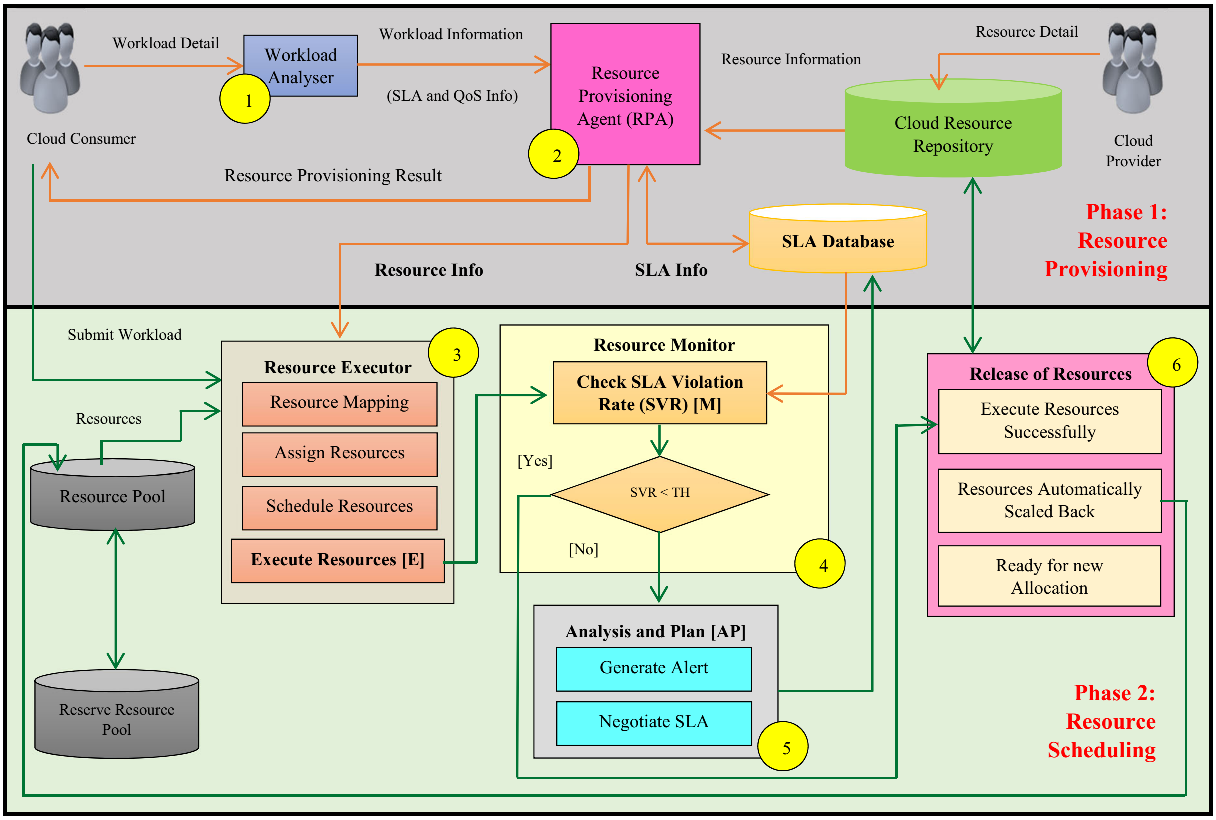


Fig. 3 Resource provisioning and resource scheduling in cloud

QoS metrics for every QoS requirement of each workload are identified [6]. Based on importance of the attribute, weight for every cloud workload is calculated [9]. After that, workloads are re-clustered based on *k-means based clustering algorithm* for better provisioning of resources. Final set of workloads is

Table 1 Pattern based clustering of workloads and their QoS requirements

Type of Workload	QoS Requirements
Web Apps	Reliable storage, High network bandwidth, High availability
Technical Computing	Computing capacity, Reliable storage
Endeavor Software	Security, High availability, Customer Confidence Level, Correctness
Performance Testing	Execution time, energy consumption and execution cost
Online Transaction Processing	Security, High availability, Internet accessibility, Usability
E-Com	Variable computing load, Customizability
Central Financial Services	Security, High availability, Changeability, Integrity
Storage and Backup Services	Reliability, Persistence
Productivity Applications	Network bandwidth, Latency, Data backup, Security
Software/Project Development and Testing	User self-service rate, Flexibility, Testing time
Graphics Oriented	Network bandwidth, Latency, Data backup, Visibility
Critical Internet Applications	High availability, Serviceability, Usability
Mobile Computing Services	High availability, Reliability, Portability

Table 2 Final set of clustered workloads

Cluster	Cluster Name	Workloads
C1	Compute	Technical Computing, Performance Testing
C2	Storage	E-Com and Storage and Backup Services
C3	Communication	Web Apps, Critical Internet Applications, Mobile Computing Services
C4	Administration	Endeavor Software, Online Transaction Processing, Central Financial Services, Productivity Applications, Software/Project Development and Testing and Graphics Oriented

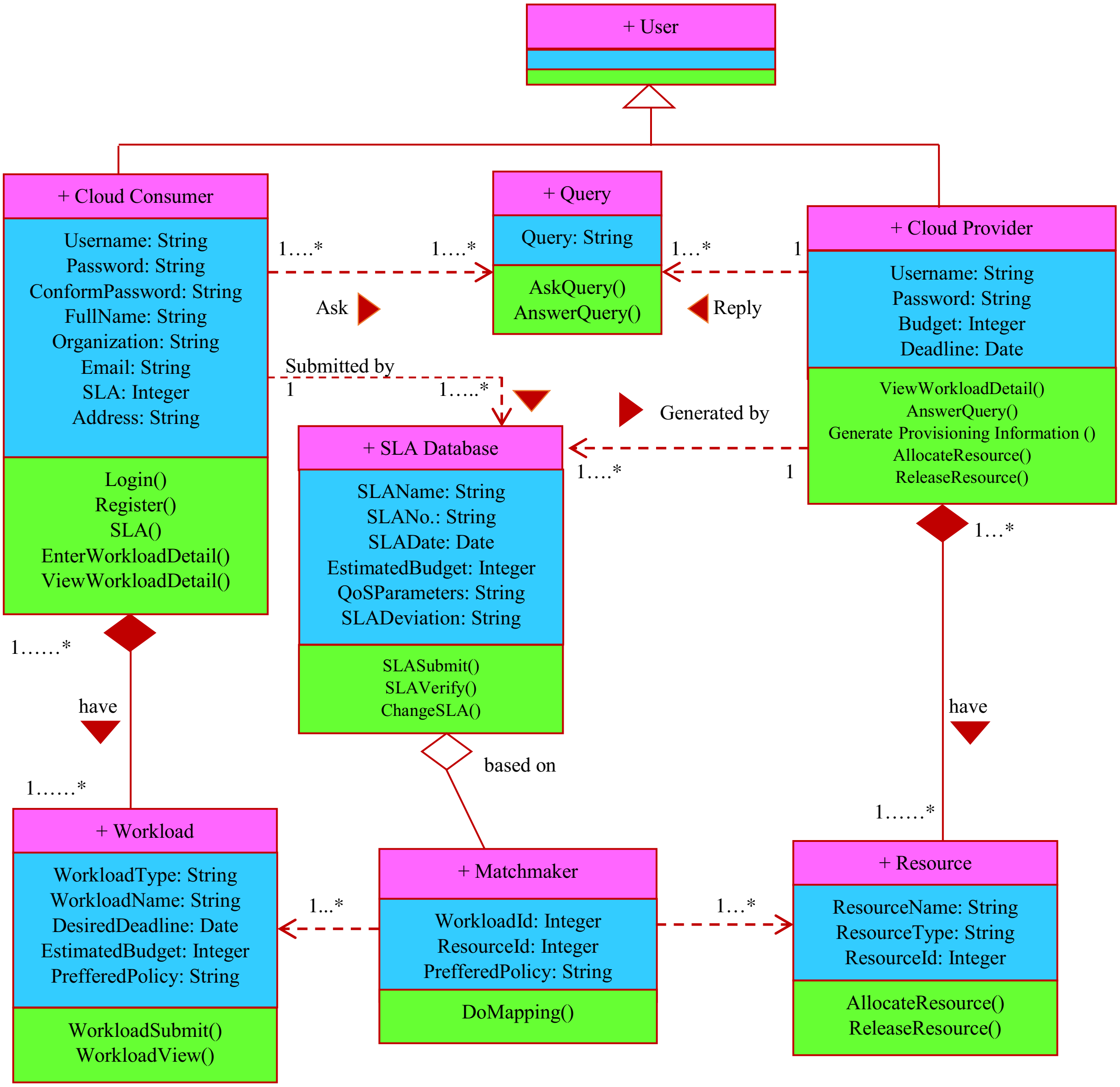


Fig. 4 Class diagram describes different interactions in SCOOTER

distributed among four clusters (C1, C2, C3 and C4) as shown in Table 2.

Resource Provisioning Agent (RPA) uses the final set of clustered workloads as an input, it accesses the cloud resource repository, which contains the information about all the resources given by resource provider and obtains the result based on QoS requirements of workload as specified by user in SLA. *QoS Manager* will calculate the execution time of workload and the expected execution time of the workloads can be derived from workload task length or historical trace data [18]. If the execution time is lesser than the desired deadline, then RPA considered workload as an urgent and put workload into Critical Queue. Further, *QoS manager* assigns priority to those workloads based on SLA information for provisioning of cloud resources. On the other hand, if the workload is not urgent then RPA put workload into non-Critical Queue for waiting [7]. In case of urgent workloads, if the deviation is more than the allowed, then penalty will be imposed (it will provision the available reserve resources to the particular workload for compensation). SLA contain details about QoS parameters considered for a particular workload and SLA deviation for provisioning of resources without violation of SLA. SLA also specifies the SLA deviation, in which workloads are executed with SLA violation rate is lesser than its threshold value [18]. RPA sends the resource provisioning result back to cloud consumer and SLA information stored in SLA database. It provisions the demanded resources to the workload for execution. Figure 4 shows the interaction of different active classes for the process of provisioning of resources. Figure 4 contains eight classes, in which there are two types of users: *Cloud Consumer* and *Cloud Provider*.

Resources and workloads details have been presented in two different classes (*Resource* and *Workload*) along with their basic description. *Matchmaker* class is used to collect the SLA information from user through another class named “*SLA Database*”. Based on SLA, *Matchmaker* Class maps the workloads with appropriate resources and returns the provisioning information to cloud user and checks the conditions of policies simultaneously. Cloud provider and cloud consumer can interact with each other through the use of *Query* class. If user requirements fulfill and required resources are available, then SCOOTER provisions the resources immediately to cloud user otherwise RPA requests to submit the workload again

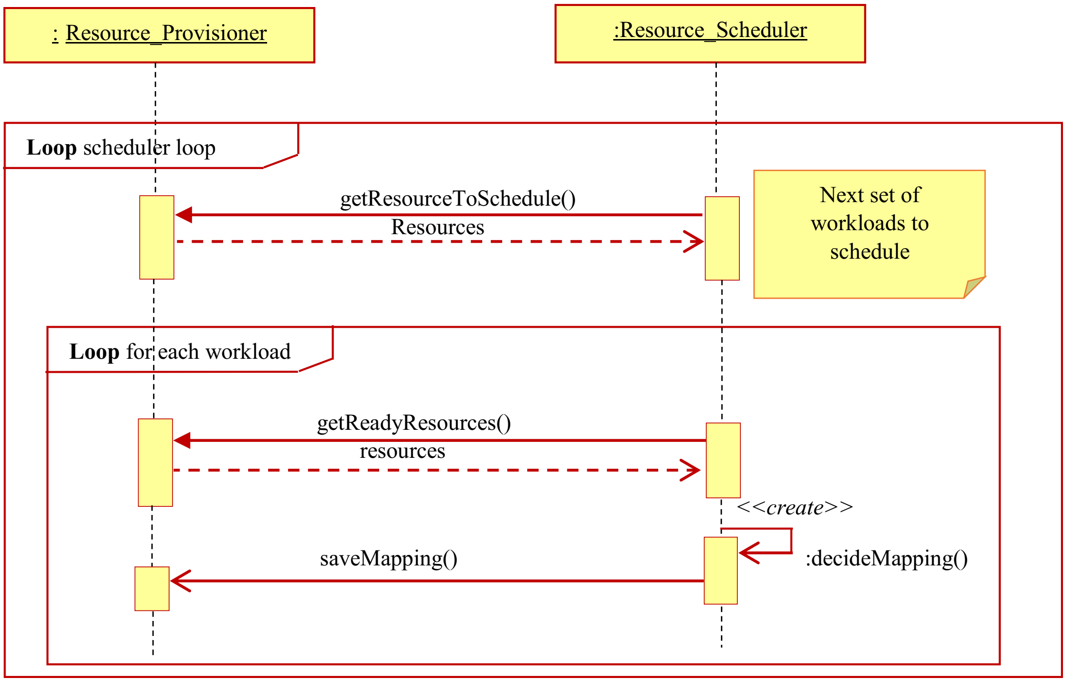


Fig. 5 Interaction between resource provisioner and resource scheduler

with new QoS requirements as a SLA document. After successfully provisioning of resources [Phase-1], workloads are submitted for resource scheduling [Phase 2].

b) Phase-2: Resource Scheduling

The objective of *Phase 2* is to schedule the provisioned resources using QRSF [11] i.e. QoS based resource provisioning technique as shown in Fig. 3. The *resource scheduler* schedules the incoming cloud workloads based on the QoS and resource details. Figure 5 shows the interaction between resource provisoner and resource scheduler for scheduling of resources. Two loops are used in scheduling of resources. First loop for scheduling i.e. get cloud resources to schedule and second loop is used to ask resources repeatedly and available resources and cloud workloads mapped efficiently based on the scheduling policies as described in Fig. 6.

In QRSF [11], four resource scheduling policies [Compromised Cost - Time Based (CCTB) Scheduling Policy, Time Based (TB) Scheduling Policy, Cost Based (CB) Scheduling Policy and Bargaining Based (BB) Scheduling Policy] have been considered based on different QoS parameters like cost, time and energy. Based on the scheduling policy, the resources are scheduled to the cloud workloads. Each workload is characterized by their deadline, estimated budget and policy. The QoS of each cloud workload is also represented in the scheduling request of the cloud workload. Figure 6 shows the decision tree based scheduling criteria, which is used to select a specific scheduling policy based on consumer workload details using predefined rules. *Note*: Pseudocode of four resource scheduling policies and their corresponding rules have been described in previous work [11, 12] respectively.

After successful scheduling of resources, *Resource Executor* executes the workload(s) on scheduled resources using selected resource scheduling policy. *Resource Executor* needs to retain the adequate number of resources for successful execution of workload during peak load. Figure 7 shows the process of resource execution and monitoring, and there are two loops used during resource execution and monitoring. First loop is scheduler loop, for computing the resource requirement for a particular cloud workload. Second loop is discovering processor properties like processing cost, speed and MIPS (Million Instructions Per Second) rating to measure resource usage. After execution of every cloud workload, the processor properties are saved for future purpose. Resource monitoring is used to monitor the performance of QoS parameters during workload execution of both physical and virtual infrastructure. The resources that are utilized by the physical and virtual infrastructures and the workloads executing on them must be measured efficiently. During execution of a particular cloud workload, *Resource Monitor (RM)* checks the status of current workload execution.

As shown in Fig. 3, during execution of cloud workloads, *performance monitor* measures the performance in terms of SLA Violation Rate to maintain the efficiency of SCOOTER and generates alert in case of performance degradation. If the problem is not resolved, then information will be send to Autonomic Service Manager for further investigation and resolution. The working of Autonomic Service Manager is described in Section 3.3.1.

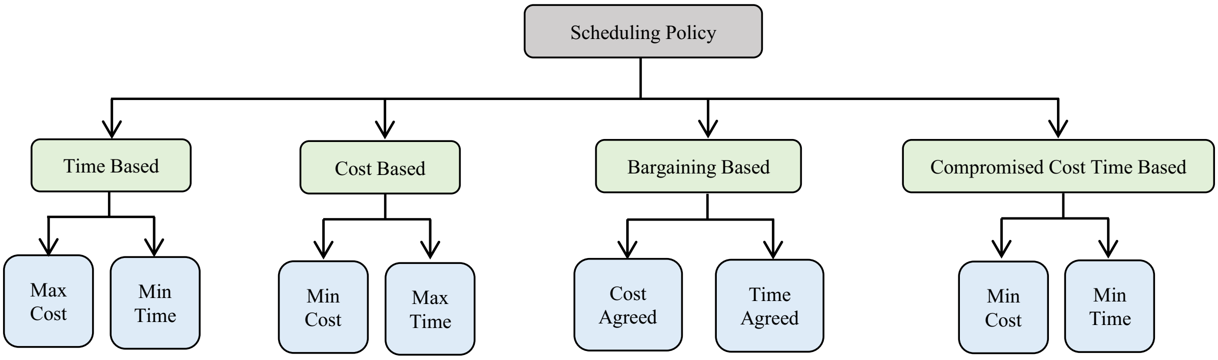


Fig. 6 Resource scheduling policies

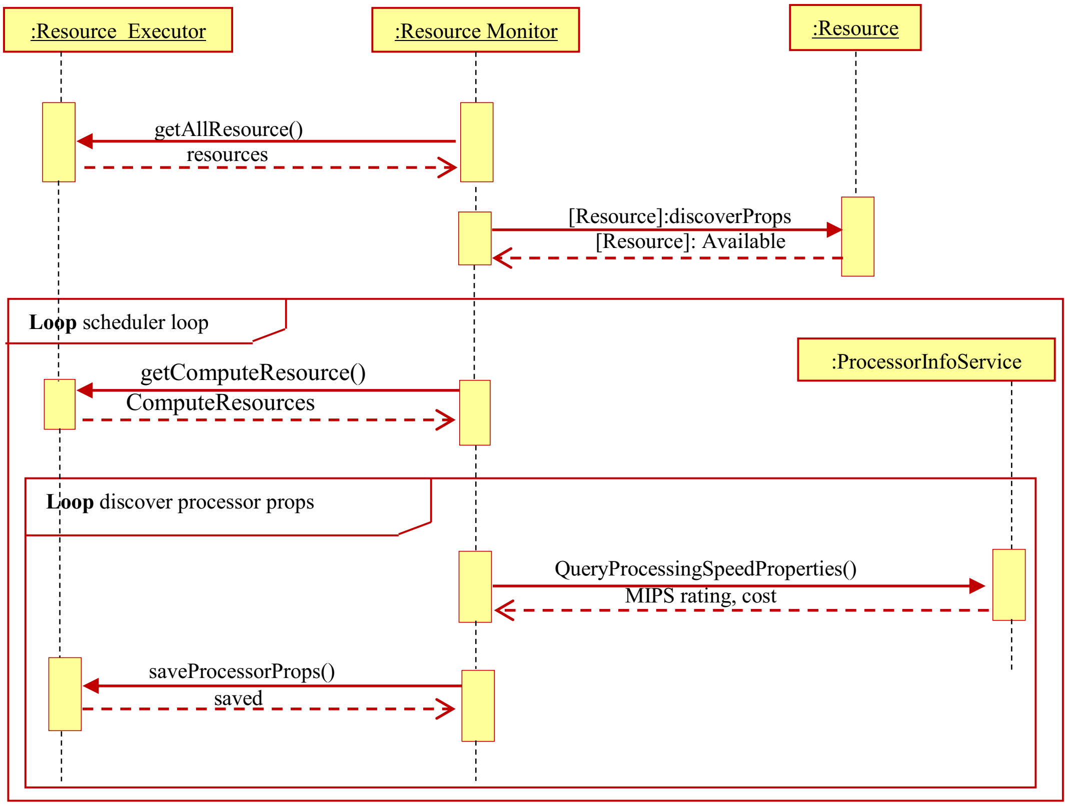


Fig. 7 Process of resource execution and monitoring

3.3.1 Autonomic Service Manager

This component is working based on IBM's autonomic model [21] that considers four steps of autonomic system: 1) Monitor, 2) Analyze, 3) Plan and 4) Execute and two external interfaces: a) Sensor and b) Effector, as shown in Fig. 1. Autonomic service manager comprises following components:

Sensors Sensors get the information about performance of current state nodes in terms of QoS parameters. Firstly, the updated information coming from processing nodes transfers to manager node then manager node transfers this information to Monitors. Updated information includes value of QoS parameters (execution time, execution cost, resource utilization, availability of service, reliability of service, energy efficiency, SLA violation rate and resource contention), faults (network, software and hardware), new updates regarding component status (outdated or missing) and security attacks (intrusion detection rate). For example: energy consumption of workload execution. Sensors continually monitor the value of energy consumption and compares with threshold value of energy consumption [49]. If the value of energy consumption is less than threshold value then continues its execution otherwise add new resources in these consecutive steps: [i) current node is declared as dead node, ii) remove dead node, iii) add new resource(s) and iv) reallocate resources and start execution] and transfers updated information to manager node.

Monitor [M] Initially, *Monitors* are used to collect the information from manager node for monitoring continuously performance variations by comparing expected and actual performance. Expected performance is the threshold value of QoS parameters, which also includes maximum value of SLA deviation and stored already in SLA Database. Actual information about performance is observed based on the failures (network, software and hardware), new updates of resources (outdated or missing), security attacks, change in QoS parameters and SLA violation, and transfers this information to next module for further analysis.

- For *self-optimizing*, QoS agent is installed on all processing nodes to monitor the performance. We have considered the set of workloads ($W_Q = \{W_1, W_2, \ldots\ldots\ldots\ldots\ldots W_m\}$) placed in workload queue and consider some or all the workloads for execution based on the availability of resources and QoS requirements of workloads. After this, resources are allocated to the workloads then Execution Time (ET), Average Cost (C) and Energy Consumption (E_{Cloud}) for every workload will be calculated. If any of the condition ([ET $\leq D_t$ && C $\leq \quad B_E$] or [$E_{Cloud} \leq E_{Threshold}$]) will be false then alert will be generated otherwise schedule resources for execution, where D_t is deadline time, B_E is allocated budget and $E_{Threshold}$ is threshold value of energy consumption.
- For *self-protecting*, security agents are installed on all the processing nodes, which are used to trace the unknown and known attacks. Based on the existing database in the system, new anomalies are captured in SCOOTER using State Vector Machine (SVM) [39]. SCOOTER captures an anomaly by detecting system intrusions and misuse of system by using its monitor and classifying it as either normal or anomalous by comparing its properties with data in SNORT database [39]. New anomalies are captured by security agent and information about anomalies is stored in database. SCOOTER protects the system from various attacks as discussed earlier such as DDoS [Smurf, LAND, SYN Flood and Teardrop], R2L [SPY, Password Guessing, IMAP], U2R [Rootkits, Buffer Overflow] and Probing [Ports sweep and NMAP]. SNORT anomaly detector is used to protect the system from attacks [39]. We have used detection engine to detect the attacks and maintain the log about attack as described in Fig. 8. Detection engine detects the pattern of every packet transferring through the network and compares with the pattern of packets existing in database to find the current value. Alert will be generated if current value is out of range [Range (Min, Max)]. SVM is used in SCOOTER to make a network profile for attack detection and it is designed based on training data [40] to detect and recognize input data (testing data) and based on the closed match to the data defined in classes, output is decided.
- For *self-healing*, software, network, hardware and hardware hardening agents are used to detect the corresponding faults. Hardware hardening agent scans drivers and checks the replica of original drivers when new node in cloud is added. After verification of new node by device driver, node is added. If the node already exists in the system, then it will generate alert. SCOOTER performs the hardening [41] of new driver into cloud to avoid the degradation of performance in case of faults and generates reports about the failure. After successful hardening of new driver, hardened driver replaces the existing drivers. If any alert is generated after hardening of driver, then original driver replaces the hardening driver and log is updated. After hardening of driver, hardware agents are using to monitor the performance of hardware components. Machine check log is used in SCOOTER to resolve hardware failures and generate alert in case of any internal error and store the information regarding alert into database. SCOOTER uses fields for Log information [*Event Type* (type of event occurred i.e. CRITICAL' OR 'ERROR'), *Event Id* (Event has unique identity number) and *Time Stamp* (Time of occurrence of error in that event)]. Database is updated by using log information [Node_Name and MAC_Address] and alert will be generated. Software agents monitor the usage of memory and CPU. SCOOTER fixes some threshold value usage for both CPU and memory [18]. If the value of usage of memory and CPU is more than threshold value, then system generates alert. Network agents are used to measure the rate of data transfer from source to destination in a particular network. SCOOTER checks the data transfer continuously; manager node asks status from processing nodes. Manager node considers network failure if node does not respond.
- For *self-configuring,* software component agent and hardware component agent are used to monitor the performance. For all the software components using at different processing nodes, status of active component is retrieved by software component agent. In SCOOTER, two types of status are defined in database: 'MISSING' or 'OUTDATED'. If software component agent shows

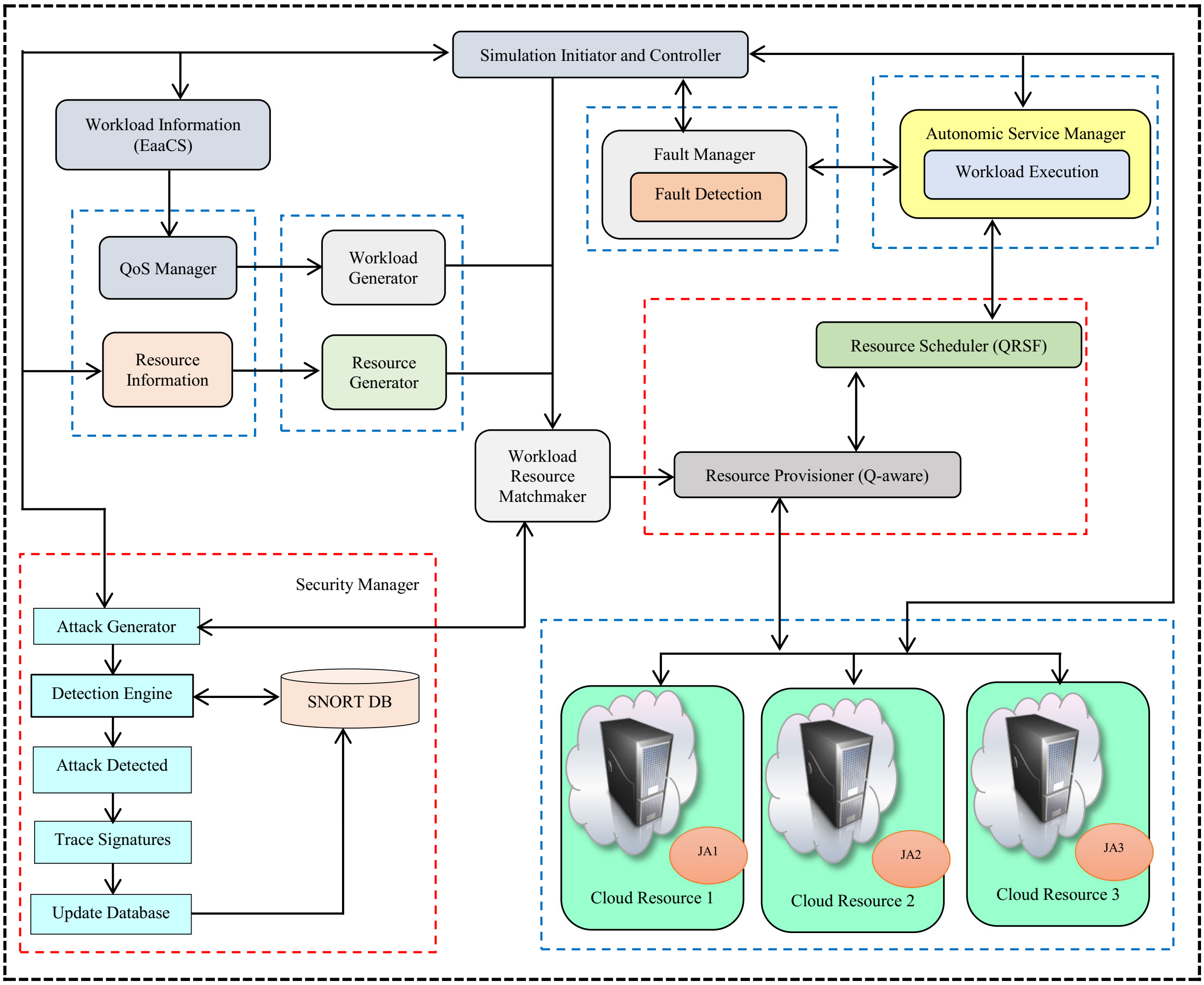


Fig. 8 Cloud testbed

status is 'MISSING' (due to missing files) then uninstall the existing software component and reinstall the component. New version of component is to be installed if the component status is 'OUTDATED'. For hardware components, SCOOTER uses fields for log information [*Event Type* (type of event occurred i.e. 'CRITICAL' OR 'ERROR'), *Event Id* (Event has unique identity number) and *Time Stamp* (Time of occurrence of error in that event)]. For all the hardware components using at different processing nodes, status of active component is retrieved by hardware component agent. If any of the event ('CRITICAL' OR 'ERROR') occurs, then database is updated by using log information [Component_Name and Compoenent_Id] and alert will be generated.

Analyze and Plan [AP] Analyze and plan module start analyzing the information received from monitoring module and make a plan for adequate actions for corresponding alert. Alerts are categorized in seven categories: QoS alert, security alert, software alert, hardware alert, network alert, software component alert and hardware component alert.

- For *self-optimizing*, the analyzing unit starts analyzing the behavior of QoS parameters of a particular node after alert is generated by QoS agent. That particular node is declared as 'DOWN' and restarts the failed node and starts it again and measures the status of that node. If the node status changes to 'ACTIVE', then continue its execution otherwise add new resources in these consecutive steps: [i) current node is declared as dead node,

ii) remove dead node, iii) add new resource(s) and iv) reallocate resources and start execution].

- For *self-protecting*, the analyzing unit starts analyzing the log information of attacks after alert is generated by security agent to generate signature. SCOOTER performs following function to generate signature: a) Collect all the new alerts generated by AE [Autonomic Element], b) Use Java utility to perform parsing to get URL, Port and Payload detail, c) Categorize data based on URL, Port and Payload, d) To find largest common substring apply LCS (Longest Common Subsequence) and e) Construct new signature by using payload string identified by LCS.
- For *self-healing*, the analyzing unit starts analyzing the behavior of hardware and software of a particular node after the alert is generated by hardware and software agent respectively. If alert is generated at runtime when workload is executing on some node N, then set the status of node N as 'DOWN' and restart the failed node and start it again and measure the status of that node. If the node status changes to 'ACTIVE', then continue its execution otherwise use another stable node after resubmission of workload. Stability of node is more if lesser number of alerts generated in past are reported from log, chance of selection of that node is more in case of failure. If workloads takes more time to execute or usage of CPU or memory is more than threshold value at a particular node then i) set the status of that node as 'DOWN', ii) restart the node, iii) identify the problem and iv) perform verification to check whether the problem is resolved or not. Network agent identifies the current status of network and to reduce failure rate, network agent takes right decision based on network log.
- For *self-configuring*, the analyzing unit starts analyzing the behavior of hardware and software component of a particular node after the alert is generated by hardware and software component respectively. If the status of hardware component is 'CRITICAL' OR 'ERROR', then declare that component as 'DOWN' and restart the failed component and start it again and measure the status of that component. If the component status changes to 'ACTIVE', then continue its execution otherwise add new component in these consecutive steps: [i) current component is declared as INACTIVE, ii) remove INACTIVE component, iii) add new component (s) and iv) start execution]. If the status of hardware component is [Event Type is 'MISSING' or 'OUTDATED')], then use following steps: i) replace the component with updated version if Event Type is 'OUTDATED' and ii) reinstall the component if Event Type is 'MISSING'. Once data has been analyzed then this framework executes the actions corresponding to the alerts automatically.

Executor [E] Executor implements the plan after analysis of current status of system.

- For *self-optimizing*, main goal of executor is to optimize the performance of QoS parameters and execute the workloads without degradation in resource utilization. Based on the information provided by analyzer, executor will add new node from resource pool with minimum execution time, cost and energy consumption. If the resources are not available in resource pool then add new node from reserve resource pool with minimum execution time, cost and energy consumption after negotiating SLA by intimating user.
- For *self-healing*, if the selected node is not a stable node then select another different node which has maximum stability among the available nodes. If the error occurred during workload execution, then save the state of that workload and restart the node.
- For *self-protecting*, SNORT [39] is used to refine the signature received from analyzer and compares new signatures with existing signature in SNORT database. If signatures are new, then they are added to SNORT database and if signatures are existing then they are merged.
- For *self-configuring*, if the new component is added then bind component by exchange messages with other existing components and start execution on that component. If the component executes the workload with minimum execution time, cost and energy consumption as required then continue execution otherwise replace with another qualified component. If error is generated in existing component, then save the state of execution and restart the component. If still component is not performing as required, then reinstall the component or install an updated version to resolve issue.

Effector Effector is acting as an interface between nodes to exchange updated information and it is used to transfer the new policies, rules and alerts to other nodes with updated information. *Note*: Pseudocode of monitoring unit, analyzing and planning unit and executing unit is described in previous work [18]. The procedure of designing new rules using fuzzy logic is described in previous work [15].

After successful execution of cloud workloads, releases the free resources to resource pool and scheduler is ready for execution of new cloud workloads.

4 Experimental Setup and Results

Figure 8 shows the cloud testbed, which is used to evaluate the performance of SCOOTER. We modeled and simulated a cloud environment using CloudSim toolkit [42]. JADE Platform [50] establishes the communication among different autonomic elements deployed at different systems using JADE Agents (JA). To measure the value of Intrusion Detection Rate, Security manager utilizes the services of SNORT with required modifications. SNORT [39] is signature-based detector and it works on Internet Protocol Networks to examine the real-time network for identification of malicious activity. It generates the "analysis signatures" by comparing with already stored signature in SNORT database and further it is refined, finalized and stores as new signatures in SNORT database. While, SVM [40] is used to detect the abnormal behavior (unknown attacks). The different tools (NMAP for probing, NetCat for L2R, Hydra for R2L and metasploit for DDoS) are used in this research work to launch different attacks [39, 40]. The concept of Carburizer [41] is used to harden the device drivers are used in Fault Manager to manage different types of software, network and hardware faults. *Note*: The detailed description of experimental setup is described in previous research work [18, 19].

We simulated computing nodes that resembles configuration of resources shown in Table 3. The execution cost is calculated based on user workload and deadline (if deadline is too early (urgent) it will be expensive because we need a greater processing speed and free resources to process particular workload with urgency. Their individual price is fixed (artificially) for different resources because all the resources are working in coordination manner to fulfill the demand of user (demand of user is changing dynamically). Experiment setup using 3 servers in which further virtual nodes ($12 = 6$ (Server 1) $+4$ (Server 2) $+2$ (Server 3)) are created. Every virtual node has different number of Execution Components (ECs) to process user workload and every EC has their own cost (C$/EC time unit (Sec)). Table 3 shows the characteristics of the resources used and their Execution Component (EC) access cost per time unit in Cloud dollars (C$) and access cost in C$ is manually assigned for experimental purposes. The access cost of an EC in C$/time unit does not necessarily reflect the cost of execution when ECs have different capabilities. The execution agent needs to translate the access cost into the C$ for each resource. Such translation helps in identifying the relative cost of resources for executing user workloads on them.

The workload is modeled as processing of user requests coming from different users for execution through as case study of *E-Commerce as a Cloud Service (EaaCS)*, which reflects the management of resources similar to cloud environment. In this outcome, we suppose that each cloud workload which is admitted to the SCOOTER may need fluctuating input size and execution time of workload and such type of cloud workloads in the form of Cloudlets are described. To validate SCOOTER, a case study of EaaCS is used, which also requires autonomic management of cloud resources [43] for execution of user requests to optimize its performance.

Table 3 Configuration details

Resource_Id	Configuration	Specifications	Operating System	Number of Virtual Node	Number of ECs	Price (C$/EC time unit)
R1	Intel Core 2 Duo - 2.4 GHz	1 GB RAM and 160 GB HDD	Windows	6	18	2
R2	Intel Core i5-2310- 2.9GHz	1 GB RAM and 160 GB HDD	Linux	4	12	3
R3	Intel XEON E 52407-2.2 GHz	2 GB RAM and 320 GB HDD	Linux	2	6	4

4.1 Case Study: E-commerce as a Cloud Service (EaaCS)

EaaCS easily adapts and scales to the unique selling and buying scenarios for your specific user needs [43]. EaaCS offers personalized, engaging and consistent service and shopping experience on any device like laptops, tabs, mobiles etc. Enable multichannel e-commerce as a service for B2C (Business to Customer) and B2B (Business to Business) businesses by seamlessly connecting e-commerce with order management within your device to include customer service, inventory, merchandizing, marketing, and financials. To get an e-commerce as a service on your handheld device/system, link your handheld device/system directly to EaaCS operational business systems with a single cloud-based platform designed specifically to integrate seamlessly with your handheld device/system.

Leading e-commerce providers have built large and complicated systems to provide countrywide or even worldwide services. However, there have been few substantive studies on e-commerce systems in real world. With the technological advances on Web, cloud computing, and mobile Internet, electronic commerce (e-commerce) becomes increasingly popular in recent years. It is estimated that in 2015, the retail e-commerce sales worldwide amounted to $1:67 trillion, and in India, 11:1% of the retail sales were on the Internet [43–45]. For enabling large volume of online transactions and providing countrywide or even worldwide services, leading e-commerce providers such as Amazon and Alibaba have built large and complicated systems. In e-commerce, service availability and system's performance are critical to providers, as it is estimated that for a leading e-commerce Website like Amazon, one second of service latency is worth tens of thousands of US dollars. However, there have been few substantive studies on large-scale e-commerce systems in real world [44].

In this paper, we select Flipkart [44] and Snapdeal [45], which are the top-two most popular e-commerce websites in India, and investigate their systems with a measurement approach [46, 47]. We analyze the workloads upon Flipkart and Snapdeal that are collected from campus network of Thapar University, India, and investigate the behaviors and performances of the e-commerce infrastructures with passive and active measurements. In particular, we characterize the flash crowd in the Dewali Shopping Day, which is the biggest online shopping festival in India, and evaluate the e-commerce system's performances under the flash crowd. As far as we know, this work is the first measurement study on large-scale e-commerce systems in real world. We find that the Dewali Shopping Day impose a massive workload on e-commerce systems; it is a challenging for e-commerce providers to accommodate the service requests during the flash crowd; and users have poor service experiences. We developed a peer-assisted architecture for e-commerce content delivery and a local-database strategy for e-commerce cloud service, and show that there is considerable room for an e-commerce provider to improve its service qualities.

We presented the first characterization study on the e-commerce workload, in particular, the massive flash crowd in the Dewali Shopping Day. We found that Dewali Shopping Day attracts many users, who are more willing to buy than usual, and the rush buying behaviors at the very beginning of the shopping festival impose a massive flash crowd on the e-commerce systems. We investigated the behaviors and performances of the e-commerce infrastructures, including the Content Delivery Network (CDN) and the cloud. We find that both Flipkart and Snapdeal have decent CDN throughputs, but the throughputs degrade significantly under the Dewali flash crowd, despite that several efforts, such as expanding CDN footprint and scheduling oversea servers, have been made to accommodate the massive content requests. We observe that Flipkart's ecommerce CDN adopts a proactive bandwidth throttling to provide low but guaranteed throughputs under the Dewali flash crowd, while Snapdeal still follows the best-effort way to provide service. As for the e-commerce clouds, both Flipkart and Snapdeal do not have sufficient capacities during the busy hours, and users suffer extraordinarily long latencies in their ecommerce transactions. We analytically show that our proposed framework (SCOOTER) can effectively manage workloads on e-commerce infrastructures, thus enable provider to improve its service qualities under massive flash crowd.

Flipkart and Snapdeal provide most of their services on Web. Roughly speaking, the e-commerce services can be categorized into two kinds: the content service and the cloud service. As shown in Fig. 9, the content service is provided with a content delivery network (CDN). Both Flipkart and Snapdeal build their

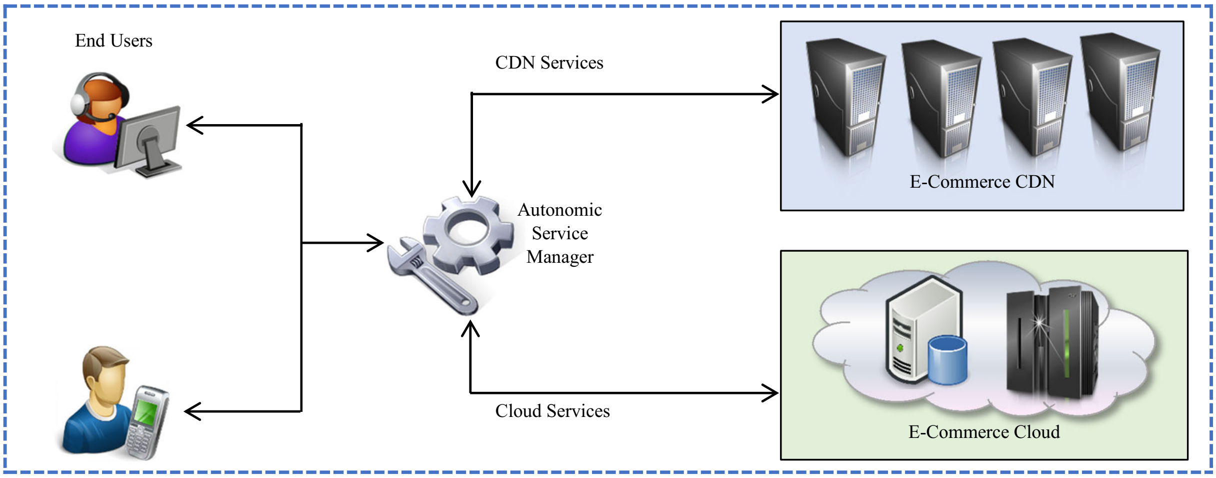


Fig. 9 E-Commerce as a cloud service

dedicated CDNs for distributing e-commerce contents, such as static Web pages, javascripts, and high-resolution images. The e-commerce CDN consists of many content servers that are deployed at locations and ISP networks that are proximate to users. Both Flipkart and Snapdeal run their private e-commerce clouds, which provide services such as search engine, recommendation, shopping cart, billing, etc. As shown in Fig. 9, an e-commerce cloud is composed of at least one cloud datacenter and many front-end Web servers. The cloud datacenter maintains elastic computing/database/storage capacities, and runs applications for all back-end jobs, such as handling users' database reads/writes/queries regarding their shopping carts, maintaining sales and inventory databases, executing ranking and recommendation algorithms, etc. The front-end server's proxy between end users and the cloud data center. More specifically, a front-end server receives a user's service request in HTTP or JSON, processes and forwards it to the cloud datacenter; when a response is returned from the datacenter, the front-end server generates a dynamic Web page containing the service response and sends it back to the user. To provide nationwide services in India, both Flipkart and Snapdeal employ DNS redirection to assign content and front-end servers to users.

In this section, we describe proposed framework in the measurement study on Flipkart and Snapdeal's e-commerce systems. For the passive measurements on Flipkart and Snapdeal, we collect traffics of the two e-commerce websites at the gateway of campus network [46], which connects tens of thousands of computers from offices, laboratories, student dormitories, etc. We employ a high-performance network traffic analyzer named iProbe to collect the e-commerce traffics [48].

For each HTTP flow, iProbe keeps a record in the log file that contains the fields such as HTTP method and URL, source/destination addresses and ports, flow size in terms of Bytes and packets in both directions, etc. With iProbe, we have collected two datasets:

- One dataset contains all the HTTP flows associated with Flipkart and Snapdeal in a week between 00:00 July 31, 2017 and 23:59 August 6, 2017. The dataset covers five weekdays and two weekends on August 5 and August 6 and refer to this dataset as WEEK.
- We also collect in 2016's Dewali Online Shopping Day from 00:00 to 23:59 on October 30, 2016. We refer to this dataset as D30. For each HTTP flow recorded by iProbe, we look up its domain name to decide which e-commerce service the flow is about. Our datasets contain all the flows between campus users and Flipkart, as all Flipkart's e-commerce traffics are on the HTTP protocol.

Unfortunately for Snapdeal, the datasets do not include all its traffics, as many Snapdeal flows are encrypted and carried on the HTTPS protocol. The performance of a case study (E-commerce as a Cloud Service) is evaluated in a cloud environment by considering QoS parameters such as execution cost, energy, execution time, SLA violation rate, fault detection rate, intrusion detection rate, resource utilization, resource contention, throughput and waiting time as discussed in next section.

4.2 Experimental Results

In this experiment work, user request considers as a cloud workload and experiment has been conducted with different number of workloads (1000-5000) for verification of performance of SCOOTER through a case study (EaaCS). The existing autonomic resource management frameworks such as EARTH [15], SRA [31], ARCS [36] and SHAPE [38] are considered to validate the SCOOTER and existing frameworks are discussed in Section 2. The various metrics used to calculate the values of different QoS parameters (execution cost, energy consumption, execution time, SLA violation rate, fault detection rate, intrusion detection rate, resource utilization, resource contention, throughput and waiting time) are described in previous research work [6, 9, 11, 12, 15–20].

Test Case 1: Execution Time - As shown in Fig. 10, the execution time increases with increase in number of workloads in SCOOTER, SHPAE, ARCS, SRA and EARTH. The average execution time in SCOOTER is 12.17% less than SHAPE, 14.76% less than ARCS, 21.69% less than SRA and 27.96% less than EARTH. After 3000 workloads, execution time increases abruptly due to increase in user workloads suddenly, but SCOOTER performs better than SHPAE, ARCS, SRA and EARTH and this is expected as the SCOOTER is keeping track of the state of all resources at each point of the time automatically which enables it to take an optimal decision than other existing frameworks.

Test Case 2: Execution Cost - With the increase in number of workloads, execution cost increases as shown in Fig. 11. As per the number of workloads increases, SCOOTER performs better than SHAPE, ARCS, SRA and EARTH. The average value of execution cost in SCOOTER is 17.25%, 18.36%, 24.35% and 28.65% less than SHAPE, ARCS, SRA and EARTH respectively. The reason is that existing frameworks does not consider the effect of other workloads in the resource scheduler at the time of workload submission but in SCOOTER, resource manager considers the effect of workloads in resource scheduler before execution of workload according to both user and resource provider's perspectives. The other reason is that with the provisioned approach (Q-aware), due to the large number of workloads, these and latter workloads had to be executed on left out resources, which may not be very cost effective.

Test Case 3: Fault Detection Rate - Figure 12 shows the capability of SCOOTER to detect the failures by injecting different number of faults in the system with different number of workloads. Fault detection rate is decreasing with increase in number of workloads. The value of fault detection rate is reducing in SCOOTER, SHAPE, ARCS, SRA and EARTH, but SCOOTER performs better. The average fault detection rate in SCOOTER is 22.66%, 24.12%, 24.98%, 27.71% more than

Fig. 10 Number of workloads vs. execution time

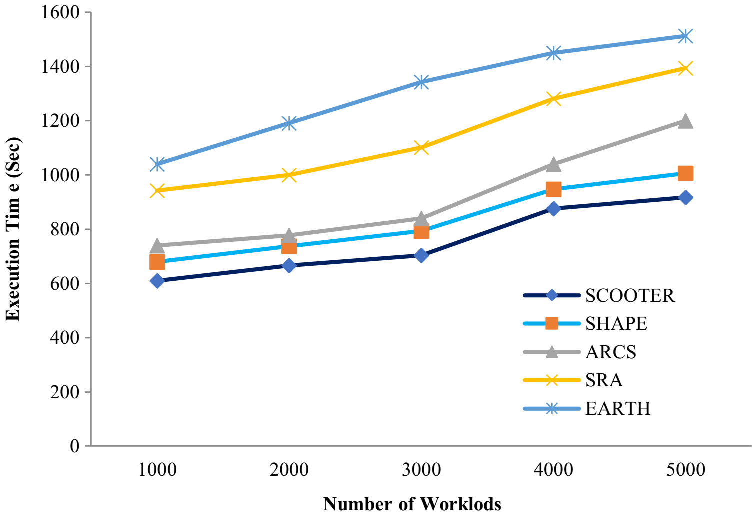
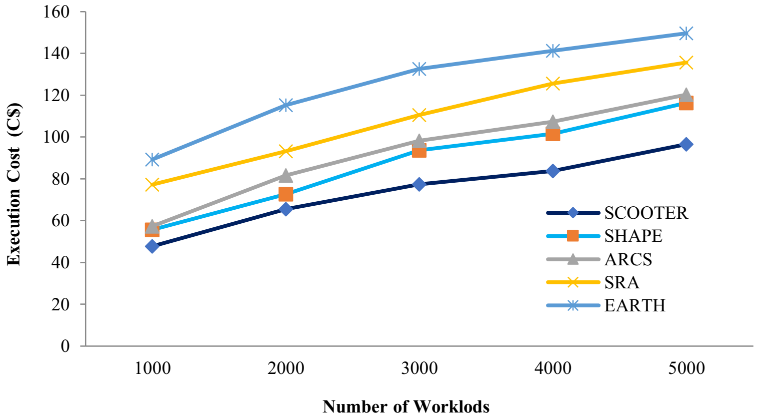


Fig. 11 Number of workloads vs. execution cost

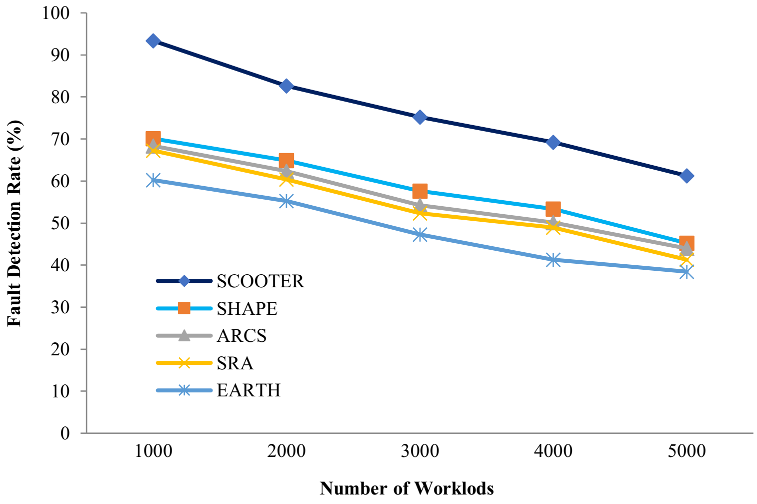


SHAPE, ARCS, SRA and EARTH respectively. SCOOTER performs effectively because it hardens the system so as to reduce the frequency of fault occurrence. It uses the concept of Carburizer [41] to harden the device drivers, in which the driver works correctly even though faults occur in the device that it controls or other faults originating outside the device.

Whenever the node is registered, the SCOOTER driver hardening agent pushes the code on that node. Once the hardening process is over, the status of nodes is forwarded to the monitor component (autonomic service manager) to prevent from future faults. This monitoring component keeps track over the proper functioning of driver automatically. If any alerts is raised because of the misbehavior of the driver, the hardened driver is replaced with the original driver and the manager node is updated to avoid same kind of future faults.

Test Case 4: Intrusion Detection Rate - For new attack or intrusion detection, database is updating with new signatures and new polices and rules are generated to avoid same attack. We have conducted experiment for known attacks (DDoS, R2L, U2R and Probing) and it is clearly shown in Fig. 13 that SCOOTER performs better than SHAPE, ARCS, SRA and EARTH. We have removed signatures of some known attacks from database to verify the effectiveness of SCOOTER. Average Intrusion Detection Rate (ITR) in SCOOTER is 16.71%, 18.22%, 21.58% and 25.26% more than SHAPE, ARCS, SRA and EARTH respectively. SCOOTER is using the SNORT anomaly detector version [39] to self-protect the system from security attacks. SNORT has been optimized to be integrated with SCOOTER. Security agents run on each node participating in the cloud and logs the details in database on Manager node of that autonomic

Fig. 12 Number of workloads vs. fault detection rate



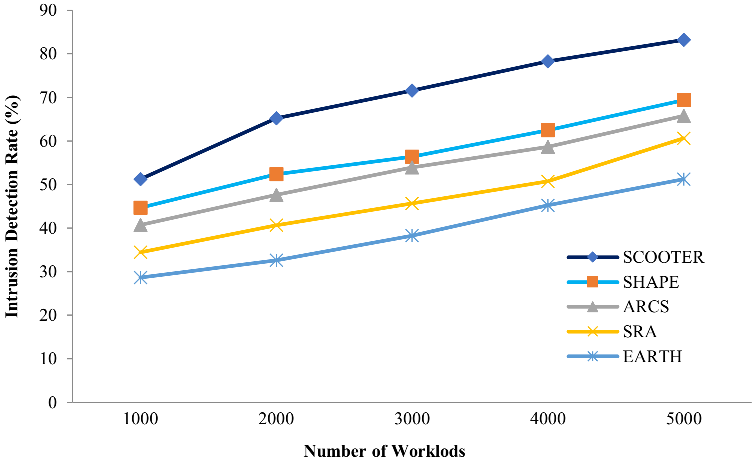


Fig. 13 Number of workloads vs. intrusion detection rate

service manager. Detection Engine in SCOOTER uses SNORT as signature based intrusion detection system to find out the signatures of known attacks in the database (SNORT DB) and uses State Vector Machine (SVM) based anomaly detector [40] to analyze the abnormal activities (unknown attacks). The training dataset is used to design SVM to find and diagnose input network traffic data to identify the attack. An action is taken once the attack is detected and it is stored into the database as shown in Fig. 8. The quick detection and removal of attack increases the intrusion detection rate.

Test Case 5: Throughput - Figure 14 shows the comparison of throughput of SCOOTER, SHAPE, ARCS, SRA and EARTH at 5000 workloads and it is clearly shown that SCOOTER performs better than SHAPE, ARCS, SRA and EARTH. In our experiment, we found the maximum value of throughput for SCOOTER at fault percentage 40%. The average throughput in SCOOTER is 13.25%, 14.18%, 18.66% and 20.36% more than SHAPE, ARCS, SRA and EARTH respectively. This is because, SCOOTER identifies the software, hardware and network faults automatically and it also prevents system from security attacks as discussed in Test Case 3 and 4, which improves the throughput of SCOOTER as compared to SHAPE, ARCS, SRA and EARTH.

Test Case 6: Waiting Time - We have injected failures to verify the performance in terms of waiting time of workloads in SCOOTER with different fault percentage (10-40%). Figure 15 shows the comparison of waiting time for SCOOTER, SHAPE, ARCS, SRA and EARTH at 5000 workloads and it is clearly shown that SCOOTER performs better. In our experiment, we found the maximum difference in waiting time with fault percentage (30%) i.e. 8.92% and at 10% fault percentage,

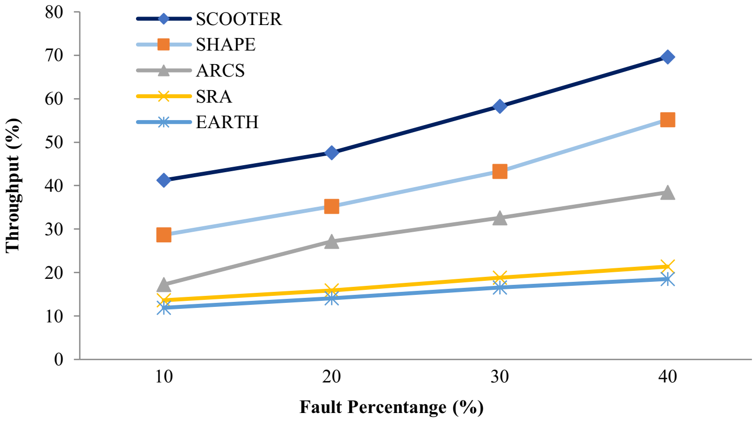
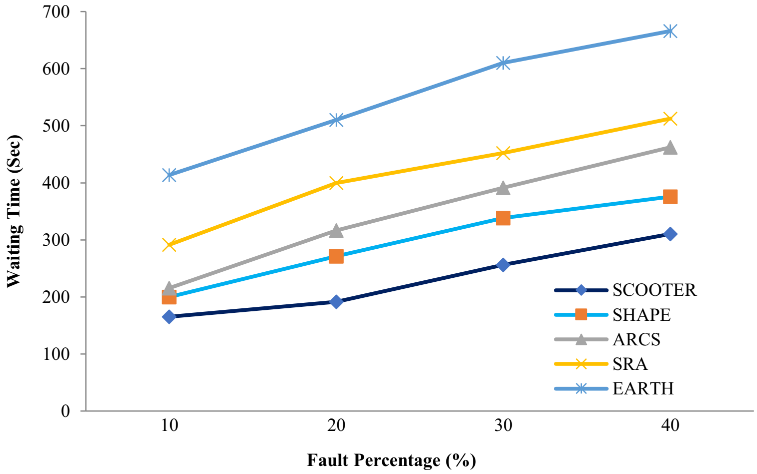


Fig. 14 Number of workloads vs. throughput [5000 Workloads]

Fig. 15 Number of workloads vs. waiting time [5000 Workloads]

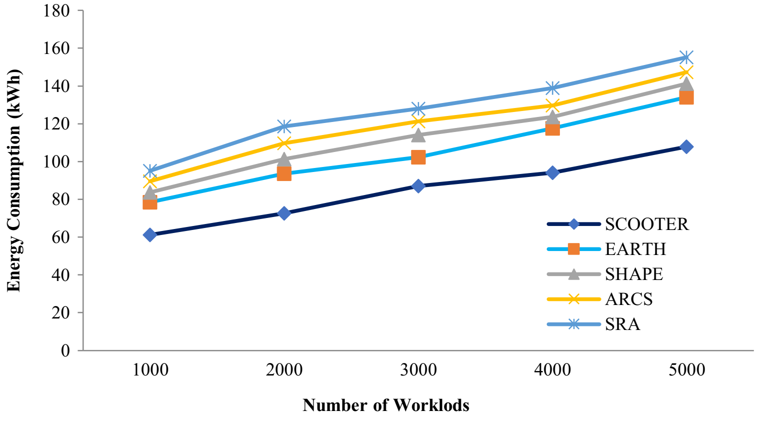


difference is just 2.71% as compared to SHAPE. Average waiting time in SCOOTER is 7.11%, 8.65%, 10.85%, 13.26% and 16.78% less than SHAPE, ARCS, SRA and EARTH respectively. The cause is that SCOOTER adjusts the provisioned resources dynamically according to the QoS requirements of workload to fulfill their required deadline, which reduces the waiting time of workload in queue. Waiting time is increasing with increase in fault percentage but SCOOTER has a capability to correct the faults automatically that also reduces waiting time.

Test Case 7: Energy Consumption - With increasing the number of cloud workloads, the value of energy consumption is increasing. The minimum value of energy consumption is 61.27 kWh at 1000 cloud workloads in SCOOTER. SCOOTER performs better than EARTH, SHAPE, ARCS and SRA in terms of energy consumption at different number of cloud workloads as shown in Fig. 16. The average energy consumption in SCOOTER is 19.87%, 23.75%, 24.65% and 28.45% less than EARTH, SHAPE, ARCS and SRA respectively. With the capability of automatically turning on and off resources according to demands, SCOOTER provisions and schedules resources efficiently and intelligently for execution of clustered workloads instead of individual workloads. Further, workload clustering reduces significant amount of network traffic due to processing similar workloads together that leads to reducing the number of active switches, which also reduces the wastage of energy.

Test Case 8: Resource Utilization - With increase in number of cloud workloads, the percentage of resource utilization is increasing. The percentage of resource utilization in SCOOTER is more than EARTH, SHAPE, ARCS and SRA at different number of workloads as shown in Fig. 17. The

Fig. 16 Number of workloads vs. energy consumption



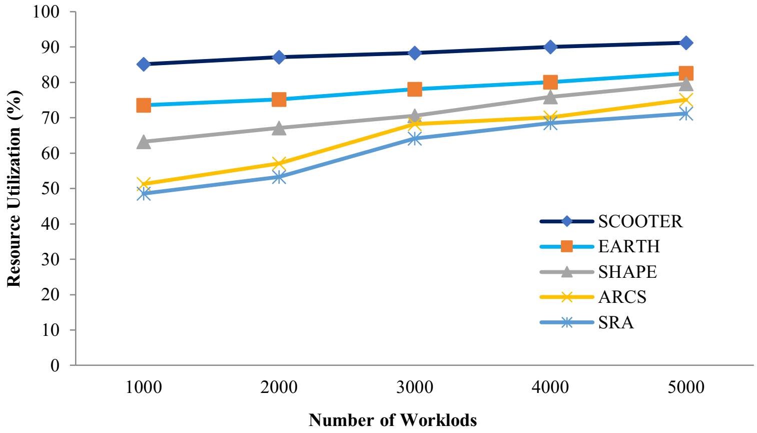


Fig. 17 Number of workloads vs. resource utilization

maximum percentage of resource utilization is 91.26% at 5000 cloud workloads in SCOOTER. The average resource utilization in SCOOTER is 8.72% more than EARTH, SHAPE, ARCS and SRA respectively. Initially, resource provisioning takes slight more time to identify the best resources based on QoS requirements of a particular workload, but later on it improves overall efficiency of resource management. Thus, the queuing time and over-utilization and under-utilization of resources will be avoided or be assuaged.

Test Case 9: SLA Violation Rate - At 1000 workloads, SLA Violation Rate (SVR) in SCOOTER is 14.39% less than SRA but SLA violation rate is suddenly decreased at 4000 workloads. SLA violation rate is in SCOOTER at 3000 workloads is 25.47% less than SRA, SHAPE, EARTH and ARCS but at 5000 workloads, SLA violation rate is 39.47% less than SRA. The SLA violation rate in SRA, SHAPE, EARTH and ARCS is more than SCOOTER as shown in Fig. 18. The average SLA violation rate in SCOOTER is 26.17%, 26.98%, 27.71% and 33.56% less than SRA, SHAPE, EARTH and ARCS respectively. This is because, SCOOTER uses admission control to reserve resources for execution of workloads in advance based on the QoS requirements specified in SLA document. Further, SCOOTER outperforms as it adjusts the resources at runtime according to the new QoS requirements of workload during its execution to avoid SLA violation.

Test Case 10: Resource Contention - We have also analyzed the effect of resource contention on number of workloads as shown in Fig. 19. With increase in number of workloads, the value of resource contention is increasing from 1000 workloads

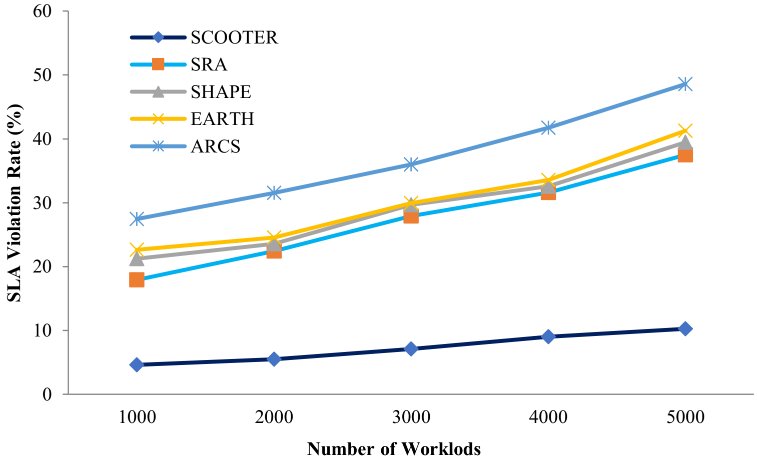


Fig. 18 Number of workloads vs. SLA violation rate

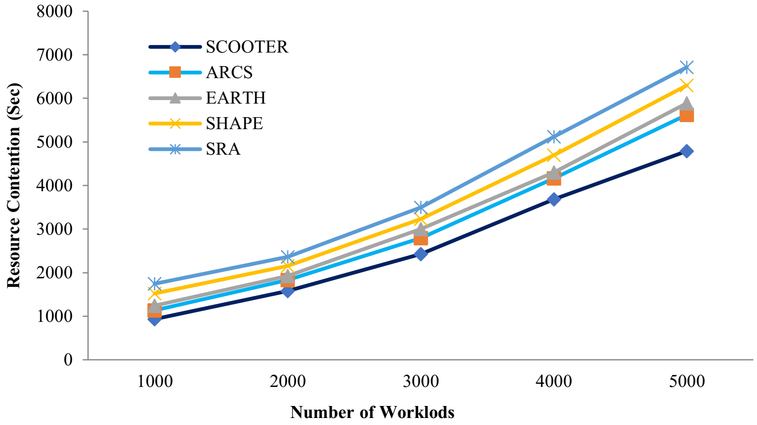


Fig. 19 Number of workloads vs. resource contention

to 5000 workloads. The value of resource contention at 1000 workloads in SCOOTER is 13.39%, 15.66%, 17.81% and 20.40% less than ARCS, EARTH, SHAPE and SRA respectively. At 5000 workloads, the value of resource contention in SCOOTER is 19.76%, 23.75%, 26.45% and 29.45% less than ARCS, EARTH, SHAPE and SRA respectively. From 1000 workloads to 4000 workloads, value of resource contention increases with same proportion in SCOOTER, ARCS, EARTH, SHAPE and SRA, but SCOOTER performs better. This is expected as the workload execution is done using SCOOTER, which is based on QoS parameters based resource provisioning policy (Q-aware). Based on deadline and priority of workload, clustering of workloads is performed, and resources are provisioned for effective scheduling. This is also because of the low variation in execution time across various resources as the resource list that is obtained from the resource provisioning unit is already filtered using Q-aware.

Every virtual node has different number for Execution Components (ECs) to process cloud workload. In this experiment, six type of ECs are considered: 6 Resources, 12 Resources, 18 Resources, 24 Resources, 30 Resources and 36 Resources to measure the variation of important QoS parameters such as execution time, execution cost and energy consumption as shown in Figs. 20, 21 and 22 respectively.

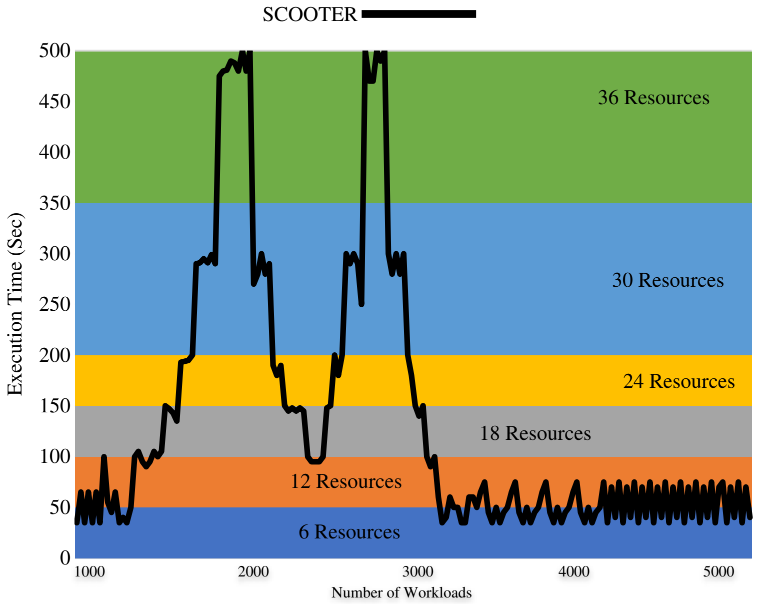
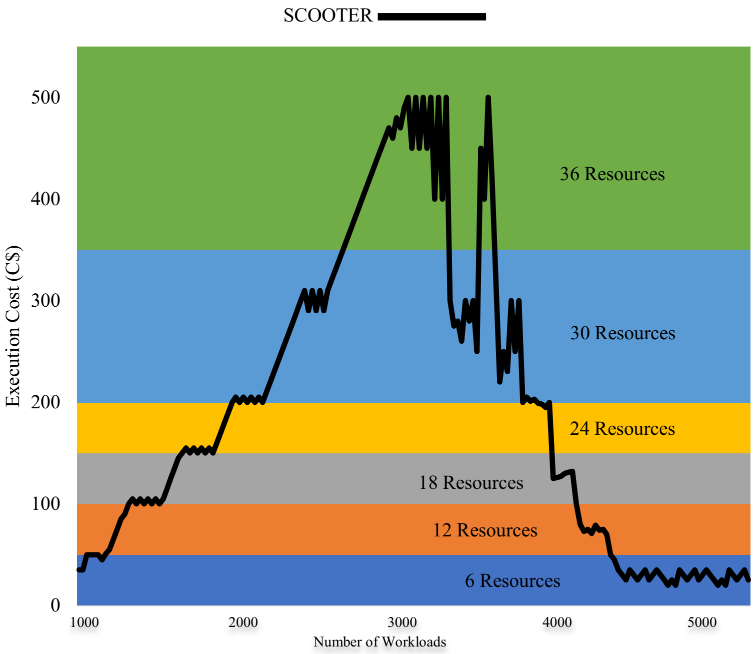


Fig. 20 Variation of execution time with number of resources

Fig. 21 Variation of execution cost with number of resources

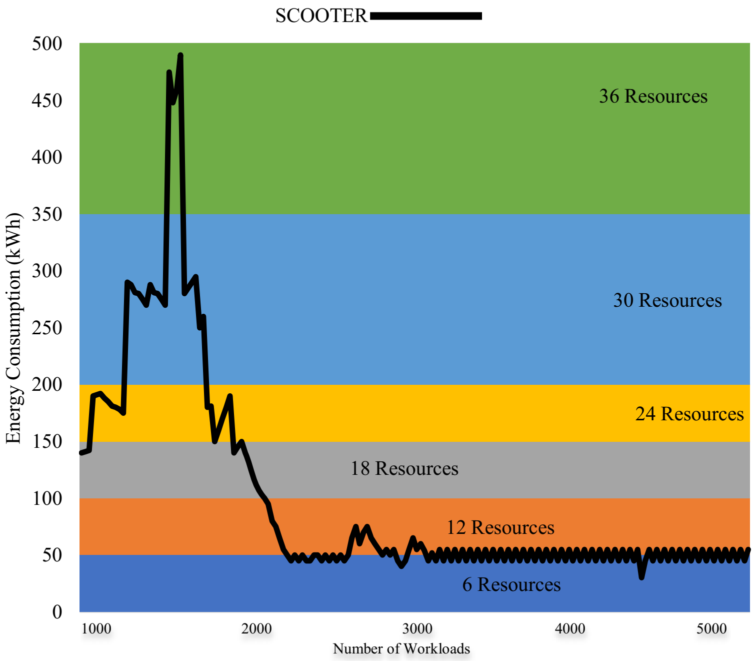


4.3 Statistical Analysis

Statistical significance of the results has been analyzed by Coefficient of Variation (*Coff. of Var.*), a statistical method [18]. Coff. of Var. is used to compare different means and furthermore offer an overall analysis of performance of the framework used for creating the statistics. It states the deviation of the data as a proportion of its average value, and is calculated as follows (2):

$$Coff.\ of\ Var. = \frac{SD}{M} \times 100 \tag{2}$$

Fig. 22 Variation of energy consumption with number of resources



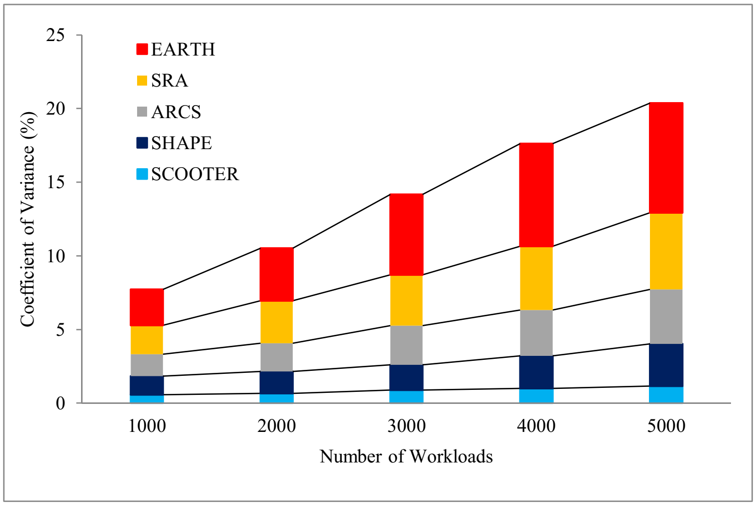


Fig. 23 Coefficient of variation for waiting time with respect to number of workloads

Where *SD* is a standard deviation and *M* is a mean. *Coff. of Var*. of waiting time of SCOOTER, SHAPE, ARCS, SRA and EARTH is shown in Fig. 23. Range of *Coff. of Var*. (0.58% - 1.16%) for waiting time approves the stability of SCOOTER.

Coff. of Var. of resource contention of SCOOTER, SHAPE, ARCS, SRA and EARTH is shown in Fig. 24. Range of *Coff. of Var*. (0.71% - 1.39%) for resource contention approves the stability of SCOOTER. Value of *Coff. of Var*. increases as the number of workloads is increasing. Small value of *Coff. of Var*. signifies SCOOTER is more efficient and stable in resource management in the situations where the number of cloud workloads are changing. SCOOTER attained the better results in the cloud for waiting time and resource contention has been studied with respect to number of workloads.

4.4 Discussions

We have verified the SCOOTER in cloud environment through a case study of e-commerce (EaaCS) and performance of SCOOTER has been compared with existing autonomic techniques (EARTH, SRA, ARCS and SHAPE) by considering different QoS parameters. Table 4 describes the comparison of QoS parameters (execution cost, execution time, energy consumption, waiting time, resource utilization, fault detection rate and throughput) used to process different number of workloads (2500 and 5000) on cloud environment for SCOOTER with different number of Virtual Machines (VMs). The number of VMs used to execute the workloads was incremented gradually showing how the QoS parameters are optimized when more VMs were added to the cloud. As shown in

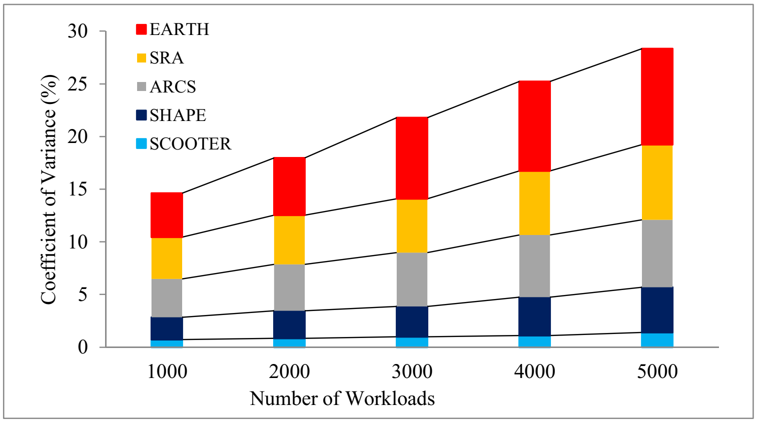


Fig. 24 Coefficient of variation for resource contention with respect to number of workloads

Table 4 Variation of QoS parameters of a bulk of cloud workloads distributed in three servers

Number of Workloads	Virtual Machines/Resources			Total Workers	Execution Time (Sec)	Execution Cost (C$)	Waiting Time (Sec)	Fault Detection Rate (%)	Throughput (%)	Resource Utilization (%)	Energy Consumption (kWh)	Resource Contention (Sec)	SVR (%)	IDR (%)
	R1	R2	R3											
2500	1	0	0	1	636.12	47.58	12.23	88.51	79.23	71.96	20.25	2522	11.27	54.92
2500	1	1	0	2	628.69	61.36	11.69	88.97	80.55	74.56	27.55	2471	11.07	56.12
2500	2	1	0	3	618.97	78.96	10.42	89.18	82.69	77.236	35.59	2237	10.59	58.69
2500	2	2	0	4	607.55	93.65	9.67	89.67	83.98	79.562	47.755	2197	10.33	60.25
2500	3	2	0	5	598.17	101.36	8.90	90.12	84.65	80.554	58.921	2143	10.03	61.65
2500	4	2	0	6	580.30	117.56	8.10	90.81	85.46	81.854	63.66	2064	9.35	63.35
2500	4	2	1	7	561.66	132.69	7.526	91.29	87.23	82.466	78.75	1927	9.02	63.93
2500	4	3	1	8	545.18	159.64	6.99	91.88	88.586	83.565	89.496	1847	8.55	65.25
2500	5	3	1	9	531.21	176.39	6.13	92.06	89.236	84.569	93.944	1768	8.13	67.18
2500	5	3	2	10	515.03	191.35	5.265	92.34	91.458	85.941	99.66	1601	7.94	69.22
2500	5	4	2	11	499.97	198.36	4.96	92.99	93.594	86.792	104.48	1498	7.71	70.12
2500	6	4	2	12	476.16	209.46	4.42	93.46	94.65	88.41	110.22	1365	7.41	71.56
5000	1	0	0	1	1403.11	322.31	17.11	82.64	76.96	78.93	140.31	4832	17.12	74.66
5000	1	1	0	2	1371.18	345.62	16.23	83.61	78.66	79.31	155.98	4512	16.95	75.12
5000	2	1	0	3	1359.66	356.21	15.63	84.22	79.12	80.33	171.545	4401	16.17	75.85
5000	3	2	0	5	1291.77	381.52	13.91	85.92	80.156	82.65	201.894	4012	14.93	76.59
5000	4	2	0	6	1268.96	396.51	12.47	86.22	81.911	83.96	206.216	3883	14.21	77.18
5000	4	2	1	7	1236.11	403.26	11.65	87.89	82.65	84.45	209.563	3571	13.55	78.01
5000	4	3	1	8	1225.19	413.56	10.65	86.52	83.695	85.25	217.45	3243	13.12	79.50
5000	5	3	1	9	1178.21	425.65	8.92	89.50	84.91	86.32	229.626	2961	12.89	80.12
5000	5	3	2	10	1151.68	447.51	8.11	90.16	85.121	87.01	241.466	2719	11.90	81.66
5000	2	2	0	4	1336.15	368.96	14.55	85.13	79.64	81.56	195.56	4236	15.25	76.26
5000	5	4	2	11	1129.11	468.95	6.62	91.68	86.568	87.78	259.566	2465	11.11	82.82
5000	6	4	2	12	1103.11	481.89	5.25	92.44	86.90	89.68	265.39	2211	10.26	83.19

Table 5 Comparison of SCOOTER with existing frameworks

Category	Framework	Mechanism	Objective Function	Merits	Future Possible Extensions
Non-Autonomic	DRMF [22]	Resource Provisioning	To improve time execution	Makespan is reduced	More QoS parameters like cost, Energy consumption etc. can be considered.
	DRP [23]		To improve execution time	Makespan is reduced	Penalty cost and compensation can be considered.
	SARP [24]		To improve relative error	Resource uptime is reduced	Failure prediction can be measured more accurately.
	DBRP [25]		To improve execution cost	Time, scalability and cost are improved	Can be extended further to add different cloud providers.
	OWS [26]	Resource Scheduling	To improve execution time	Energy consumption is reduced	Average decision time can be reduced.
	ACOJS [27]		To improve completion time	Execution time is reduced	SLA violation can be reduced.
	HEFT [28]		To improve communication time	Energy consumption and execution time are reduced	Cost can be reduced.
	PSOH [29]		To improve communication cost	Cost is reduced	Execution time can be reduced.
Autonomic	ASP [30]	Self-healing	To improve negotiation time for SLA	Reduced SLA violations and lease cost	Decision Delay can be improved
	SRA [31]		To improve execution cost and time	Execution time and network traffic are reduced	Stabilized API can be incorporated.
	CBR [32]	Self-configuring	To Reduce Execution time and cost	CPU Time and SLA Violations are reduced	More QoS parameters like cost, time etc. can be considered.
	COCCUS [33]		To reduce maintenance cost	Cost is reduced	Problem of starvation can be reduced.
	CAS [34]	Self-optimizing	To optimize resource utilization and cost	Average CPU time and Cost are reduced	Can be extended for network intensive applications.
	AWM [35]		To reduce monetary cost	Makespan is reduced	Penalty cost and compensation can be considered.
	ARCS [36]		To reduce resource contention	Time of resource contention is reduced	Execution time and cost can be reduced.
	EARTH [15]		To reduce energy consumption	Resource utilization is improved	Self-protection, Self-healing and Self-configuration can be considered.
	RASP [37]	Self-protecting	To improve security	Reduced security breaching	Self-optimization, Self-healing and Self-configuration can be considered.
	SHAPE [38]		To improves security, execution time and cost	Time and cost are reduced, and availability, reliability and security are improved	Self-optimization and Self-configuration can be considered.
	SCOOTER	Self-healing, Self-configuring, Self-optimizing and Self-protecting	To improve user satisfaction and increases reliability and availability of services.	It finds and reacts to sudden faults, optimizes QoS parameters, configure/ reconfigure resources and detects and protects from cyber-attacks automatically.	SCOOTER can be extended by developing pluggable scheduler, in which resource scheduling policy can be changed easily based on the requirements.

Table 4, with one virtual node running on Server R1, execution of 2500 workloads finished in 636.12 seconds. With 12 virtual nodes (6 running on R1, 4 running on R2 and 2 running on R3), the application took 476.16 seconds. We note that the execution time is reduced by adding additional virtual nodes.

Statistical significance of the results has been analyzed by coefficient of variation, a statistical method to measure of the distribution of data about the mean value to find the stability of SCOOTER with small value of coefficient of variation. Considering all these experimental results, it is shown that the SCOOTER delivers a superior autonomic solution for heterogeneous cloud workloads and approximate optimum solution for challenges of autonomic resource management.

4.5 Comparison of SCOOTER with Existing Frameworks

In the proposed framework, autonomic resource provisioning and scheduling has been done on the basis of different QoS parameters considered for self-healing, self-configuring, self-optimizing and self-protecting, which was not considered traditionally. Experimental results show that SCOOTER is able to schedule the resources efficiently for workload execution automatically. Table 5 shows the comparison of SCOOTER with existing frameworks based on different features (functionally-wise).

Further, the implementation of SCOOTER in a real cloud environment can be achieved using different available cloud platforms such as OpenStack, Docker based Container's Management System and Amazon EC2.

- SCOOTER can be deployed on OpenStack for effective management of cloud resources at IaaS level. Further, OpenStack based implementation of proposed framework can enhance scalability of cloud services. The deployment of SCOOTER can provision and schedule the cloud resources using different hypervisors (Xen, VMware or kernel-based virtual machine (KVM) for instance) and several virtualization technologies (such as bare metal or high-performance computing).
- Today's de facto container technology (Docker), uses resource isolation features of the Linux kernel such as cgroups and kernel namespaces to allow independent ”containers” to run within a single Linux instance, avoiding all the heavy overhead of starting VMs on hypervisors. The VMs transition based deployment of SCOOTER can be shifted to container-based deployments, which can increase realization and lower overheads associated with container deployment can be used to support real-time workloads. The deployment of SCOOTER on Docker based Container's Management System improves the performance due to following reasons: a) containers start up very quickly and their launching time is less than a second and b) containers have tiny memory footprint and consume a very small amount of resources. Compared with VM based cloud testbed, using containers not only improves the performance of SCOOTER, but also allows the host to support more instances simultaneously.
- Amazon Elastic Compute Cloud (Amazon EC2) also provides scalable computing capacity in the Amazon Web Services (AWS) cloud by utilizing its API. The deployment of SCOOTER on Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, quickly scale capacity, both up and down, as computing requirements change dynamically, which can further reduce the SLA violation rate and execution time of workload execution.

5 Conclusions and Future Work

In this paper, a resource management framework called SCOOTER has been proposed and it has an ability to manage resources automatically through properties of autonomic management, which are self-healing (find and react to sudden faults), self-optimizing (maximize resource utilization and energy efficiency and minimize execution cost, execution time, resource contention and SLA violation rate), self-configuring (capability to readjust resources) and self-protecting (detection and protection of cyber-attacks) automatically with minimum human involvement. SCOOTER has been validated in cloud environment through a case study (E-commerce as a Cloud Service) and the experimental results shows that SCOOTER performs better than existing autonomic resource management frameworks in terms of different QoS parameters. SCOOTER efficiently schedules the provisioned cloud resources automatically for execution of heterogeneous workloads and maintains the SLA which improves user satisfaction.

In future, SCOOTER can also be extended by developing pluggable scheduler, in which resource scheduling can be changed easily based on the new requirements.

Acknowledgements One of the authors, Dr. Sukhpal Singh Gill [Post Doctorate Fellow], gratefully acknowledges the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia, for awarding him the Fellowship to carry out this research work. We thank Adel Nadjaran Toosi and anonymous reviewers for their comments on improving the paper.

References

1. Varghese, B., Buyya, R.: Next generation cloud computing: New trends and research directions. Future Generation Comput. Syst. **79**, 849–861 (2017). https://doi.org/10.1016/j.future.2017.09.020
2. Qi, Z.T.L., Cheng, Z., Li, K., Khan, S.U., Li, K.: An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. J. Grid Comput. **14**(1), 55–74 (2016)
3. de Carvalho, O.A. Jr., Adilson, O., Bruschi, S.M., Santana, R.H.C., Santana, M.J.: Green cloud meta-scheduling. J. Grid Comput. **14**(1), 109–126 (2016)
4. Jiang, J., Lin, Y., Xie, G., Fu, L., Yang, J.: Time and Energy Optimization Algorithms for the Static Scheduling of Multiple Workflows in Heterogeneous Computing System. J. Grid Comput., 1–22 (2017). https://doi.org/10.1007/s10723-017-9391-5
5. Ebrahimirad, V., Goudarzi, M., Rajabi, A.: Energy-aware scheduling for precedence-constrained parallel virtual machines in virtualized data centers. J. Grid Comput. **13**(2), 233–253 (2015)
6. Singh, S., Chana, I.: Metrics based workload analysis technique for IaaS cloud. In: The Proceeding of International Conference on Next Generation Computing and Communication Technologies 23 - 24 April 2014, Dubai, pp. 1–6 (2014)
7. Chana, I., Singh, S.: Quality of service and service level agreements for cloud environments: Issues and challenges, cloud Computing-Challenges, limitations and R&D solutions, 51-72 springer international publishing (2014)
8. Singh, S., Chana, I.: Cloud resource provisioning: survey, Status and Future Research Directions. Knowl. Inf. Syst. **49**(3), 1005–1069 (2016)
9. Singh, S., Chana, I.: Q-aware: quality of service based cloud resource provisioning. Comput. Electr. Eng. **47**, 138–160 (2015)
10. Singh, S., Chana, I.: A survey on resource scheduling in cloud computing issues and challenges. J. Grid Comput. **14**(2), 217–264 (2016)
11. Singh, S., Chana, I.: QRSF Qos-aware resource scheduling framework in cloud computing. J. Supercomput. **71**(1), 241–292 (2015)
12. Singh, S., Chana, I.: Resource provisioning and scheduling in clouds: QoS perspective. J. Supercomput. **72**(3), 926–960 (2016)
13. Singh, S., Chana, I.: QoS-aware autonomic cloud computing for ICT. In: The proceeding of International Conference on Information and Communication Technology for Sustainable Development (ICT4SD - 2015), Ahmedabad, India, 3 - 4 July 2015, pp. 569–577. Springer, Singapore (2016)
14. Singh, S., Chana, I.: Qos-aware autonomic resource management in cloud computing: a systematic review. ACM Comput. Surv. **48**(3), 1–46 (2015)
15. Singh, S., Chana, I.: EARTH: energy-aware autonomic resource scheduling in cloud computing. J. Intell. Fuzzy Syst. **30**(3), 1581–1600 (2016)
16. Singh, S., Chana, I., Singh, M.: The journey of QoS-aware autonomic cloud computing. IEEE IT Professional **19**(2), 42–49 (2017)
17. Singh, S., Chana, I., Buyya, R.: STAR: SLA-aware autonomic management of cloud resources. In: IEEE Transactions on Cloud Computing, pp. 1–14 (2018). https://doi.org/10.1109/TCC.2017.2648788
18. Sukhpal S.G., Chana, I., Singh, M., Buyya, R.: CHOPPER: an Intelligent QoS-aware autonomic resource management approach for cloud computing cluster computing, pp. 1–39 (2017). https://doi.org/10.1007/s10586-017-1040-z/ Available Online: https://link.springer.com/article/10.1007/s10586-017-1040-z
19. Sukhpal S.G., Buyya, R., Chana, I., Singh, M., Abrahiam, A.: BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources, Journal of Network and Management System, pp. 1–40. Springer, Berlin (2017). https://doi.org/10.1007/s10922-017-9419-y
20. Singh, S., Chana, I., Singh, M., Rajkumar, B.: SOCCER self-optimization Of energy-efficient cloud resources. Clust. Comput. **19**(4), 1787–1800 (2016)
21. Kephart, J.O., Walsh, W.E.: An architectural blueprint for autonomic computing. Technical Report, IBM Corporation, 1-29, IBM. http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf (2003)
22. Quiroz, A., Kim, H., Parashar, M., Gnanasambandam, N., Sharma, N.: Towards autonomic workload provisioning for enterprise grids and clouds. In: 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 50–57. IEEE (2009)
23. Vecchiola, C., Calheiros, R.N., Karunamoorthy, D., Buyya, R.: Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. Futur. Gener. Comput. Syst. **28**(1), 58–65 (2012)
24. Herbst, N.R., Huber, N., Kounev, S., Amrehn, E.: Self-adaptive workload classification and forecasting for proactive resource provisioning. Concurrency Comput.: Pract. Exp. **26**(12), 2053–2078 (2014)
25. Qavami, H.R., Jamali, S., Akbari, M.K., Javadi, B.: Dynamic resource provisioning in cloud computing: a heuristic markovian approach. In: Cloud Computing, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering,

vol. 133, pp. 102–111. Springer International Publishing (2014)

26. Varalakshmi, P., Ramaswamy, A., Balasubramanian, A., Vijaykumar, P.: An optimal workflow based scheduling and resource allocation in cloud. In: Advances in computing and communications, pp. 411–420. Springer, Berlin (2011)
27. Li, K., Gaochao, X., Zhao, G., Dong, Y., Wang, D.: Cloud task scheduling based on load balancing ant colony optimization. In: Sixth Annual Chinagrid Conference (ChinaGrid), pp. 3–9. IEEE (2011)
28. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Task scheduling algorithms for heterogeneous processors. In: Proceedings of the Eighth Heterogeneous Computing Workshop (HCW'99), pp. 3–14. IEEE (1999)
29. Pandey, S., Wu, L., Guru, S., Buyya R: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), Perth, Australia (2010)
30. Cardellini, V., Casalicchio, E., Presti, F.L., Silvestri, L.: SLA-aware resource management for application service providers in the cloud. In: First International Symposium on Network Cloud Computing and Applications (NCCA), pp. 20–27. IEEE (2011)
31. Wu, L., Garg, S.K., Buyya, R.: SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments. In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 195–204. IEEE (2011)
32. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for cloud infrastructure management. Futur. Gener. Comput. Syst. **29**(2), 472–487 (2013)
33. Konstantinou, I., Kantere, V., Tsoumakos, D., Koziris, N.: COCCUS: self-configured cost-based query services in the cloud. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1041–1044. ACM (2013)
34. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: 2010 11th IEEE/ACM International Conference on In Grid Computing (GRID), pp. 41–48. IEEE (2010)
35. Sah, S.K., Joshi, S.R.: Scalability of efficient and dynamic workload distribution in autonomic cloud computing. In: International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), pp. 12–18. IEEE (2014)
36. Sheikhalishahi, M., Grandinetti, L., Wallace, R.M., Vazquez-Poletti, J.L.: Autonomic resource contention-aware scheduling. Softw.: Pract. Exp. **45**(2), 161–175 (2015)
37. Yuan, E., Malek, S., Schmerl, B., Garlan, D., Gennari, J.: Architecture-based self-protecting software systems. In: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, pp. 33–42. ACM (2013)
38. Chopra, I., Singh, M.: SHAPE—An approach for self-healing and self-protection in complex distributed networks. J. Supercomput. **67**(2), 585–613 (2014)
39. Caswell, B., Beale, J.: Snort 2.1 intrusion detection, Syngress (2004)
40. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pp. 144–152. ACM (1992)
41. Kadav, A., Renzelmann, M.J., Swift, M.M.: Tolerating hardware device failures in software. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, pp. 59–72. ACM (2009)
42. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw.: Pract. Exp. **41**(1), 23–50 (2011)
43. Talib, A.M., Alomary, F.O.: Cloud computing based E-Commerce as a service model: impacts and recommendations. In: Proceedings of the International Conference on Internet of Things and Cloud Computing, p. 27. ACM (2016)
44. Prasad, C.S.D., Rao, S.R.S.: Competition in the indian E-Commerce sector durga prasad the case of flipkart. Gavesana J. Manag. **7**(2), 1–22 (2015)
45. Chauhan, P.: A Comparative study on consumer Preferences towards online retail marketers-with special reference to Flipkart, Jabong, Amazon, Snapdeal Myntra and fashion and you. IJAR **1**(10), 1021–1026 (2015)
46. Sebastian, M., Jercinovic, S., Cosmina, T., Simonacarmen, D., Cosmin, S.: A study regarding online traffic analytics of websites for profit. Agricultural Management/Lucrari Stiintifice Seria I. Manag. Agricol **19**(1), 81–84 (2017)
47. Luo, J., Liang, Y., Gao, W., Yang, J.: Hadoop based deep packet inspection system for traffic analysis of e-business websites. In: International Conference on Data Science and Advanced Analytics (DSAA), pp. 361–366. IEEE (2014)
48. Arora, N., Zhang, H., Rhee, J., Yoshihira, K., iProbe, G.J.: A lightweight user-level dynamic instrumentation tool. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, pp. 742–745. IEEE Press (2013)
49. Sukhpal S.G., Buyya, R.: A taxonomy and future directions for sustainable cloud computing: 360 degree view. http://www.buyya.com/papers/SustainableClouds360.pdf
50. Exposito, J.A., Ametller, J., Robles, S.: Configuring the JADE HTTP MTP. http://jade.tilab.com/documentation/tutorials-guides/configuring-the-jade-http-mtp/ (2010)