Cluster Comput (2018) 21:1203–1241
https://doi.org/10.1007/s10586-017-1040-z

CHOPPER: an intelligent QoS-aware autonomic resource management approach for cloud computing

Sukhpal Singh Gill[1] · Inderveer Chana[2] · Maninder Singh[2] · Rajkumar Buyya[1]

Received: 2 July 2016 / Revised: 31 May 2017 / Accepted: 10 July 2017 / Published online: 1 August 2017

Abstract Cloud computing is the future generation of computational services delivered over the Internet. As cloud infrastructure expands, resource management in such a large heterogeneous and distributed environment is a challenging task. In a cloud environment, uncertainty and dispersion of resources encounters problems of allocation of resources. Unfortunately, existing resource management techniques, frameworks and mechanisms are insufficient to handle these environments, applications and resource behaviors. To provide an efficient performance and to execute workloads, there is a need of quality of service (QoS) based autonomic resource management approach which manages resources automatically and provides reliable, secure and cost efficient cloud services. In this paper, we present an intelligent QoS-aware autonomic resource management approach named as CHOPPER (**C**onfiguring, **H**ealing, **O**ptimizing and **P**rotecting **P**olicy for **E**fficient **R**esource management). CHOPPER 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. We have evaluated the performance of the proposed approach in a real cloud environment and the experimental results show that the proposed approach performs better in terms of cost, execution time, SLA violation, resource contention and also provides security against attacks.

Keywords Autonomic cloud computing · Resource provisioning and scheduling · Self-healing · Self-configuring · Self-optimizing · Self-protecting

1 Introduction

Cloud computing offers pay per use based services such as infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) through different cloud providers [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. Currently, cloud services are provisioned and scheduled according to resources' availability without ensuring the expected performances [2]. The cloud provider should evolve its ecosystem in order to meet QoS-aware requirements of 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 [3]. 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 etc. Autonomic management implies the fact that the service is able to self-manage itself as per

✉ Sukhpal Singh Gill
ssgill@thapar.edu

Inderveer Chana
inderveer@thapar.edu

Maninder Singh
msingh@thapar.edu

Rajkumar Buyya
rbuyya@unimelb.edu.au

[1] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC, Australia

[2] Computer Science and Engineering Department, Thapar University, Patiala, Punjab, India

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 [4]. Based on human guidance, an intelligent autonomic system keeps the system stable in unpredictable conditions and adapts quickly in new environmental conditions like software, hardware failures etc. 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 [5]. Autonomic cloud computing system is based on MAPE-K loop [6] 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 of self-management [1,41]:

- Self-healing is a capability of an intelligent autonomic system to identify, analyze and recover from unfortunate faults automatically.
- Self-configuring is a capability of an intelligent autonomic system to adapt to the changes in the environment.
- Self-optimizing is a capability of an intelligent autonomic system to improve the performance.
- Self-protecting is a capability of an intelligent autonomic system to protect against intrusions and threats.

In our earlier work, QoS based resource provisioning and scheduling (QRPS) framework [33] is proposed. 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. To design a QoS based autonomic resource management approach, QRPS has been further extended by proposing energy-aware autonomic resource scheduling technique (EARTH), in which MAPE-K loop 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 [6]. 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 [36] is proposed which clusters the heterogeneous cloud workloads and executes them with minimum energy consumption, but SOCCER only focuses on one aspect of self-optimization i.e. energy consumption. In this research work, QoS-aware autonomic resource management approach (CHOPPER) 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.

The motivation of this paper is to design an intelligent cloud based and QoS-aware autonomic resource management approach called CHOPPER (**C**onfiguring, **H**ealing, **O**ptimizing and **P**rotecting **P**olicy for **E**fficient **R**esource management). CHOPPER 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. CHOPPER manages resources automatically and offers 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). The performance of CHOPPER is tested in a real cloud environment. CHOPPER improves user satisfaction and increases reliability and availability of services. The rest of the paper is organized as follows. Section 2 presents related work and contributions. Proposed approach is presented in Sect. 3. Section 4 describes the experimental setup and present experimental results. Section 5 presents conclusions and future scope.

2 Related work

Self-management of resources is the soul of autonomic cloud computing. Management of resources in cloud has been done through different techniques in the existing literature but autonomic management of resources is challenging. The current research work done in the area of self-management is described in this section.

2.1 Existing QoS-aware autonomic resource management techniques

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.

2.1.1 Self-healing

In cloud computing, self-healing is a capability of a system to identify, analyze and recover from unfortunate faults automatically. Application service provider (ASP) [8] uses web service description language (WSDL) 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. In this system, lease cost and SLA violations are reduced but consumes very large execution time. Self-healing SLA (SH-SLA) [9] is an autonomic system designed to enable hierarchical self-healing which monitors SLA, SLA violation and takes necessary steps to prevent SLA violations. SLAs with similar agreement interact with each other to notify the status of execution but SH-SLA is not able to optimize the required execution time and cost. Self-organizing and healing (SNOOZE) [10] uses hierarchical architecture to allocate the resources to the workloads in a virtual environment. SNOOZE is working in three layers: physical layer, hierarchical layer and client layer. Local controller is used to control the nodes and machines are structured in clusters in physical layer. Design of SNOOZE is very simple but scalability is achieved without considering execution time and cost. Adaptive fault tolerance in real time cloud (AFRTC) [11] system is used to detect the faults, provide fault-tolerance and to calculate the reliability of nodes to take decisions. Reliability of nodes in virtual environment is changing adaptively. Node is reliable if a virtual node produces results within specified deadline otherwise node is not reliable. Due to addition of resource(s) in case of resource failure, cost of AFRTC is increased which leads to customer dissatisfaction.

2.1.2 Self-configuring

In cloud computing, self-configuring is a capability of a system to adapt to the changes in the environment. Case base reasoning (CBR) [12] 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. 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) [13] uses 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. Autonomic resource allocation strategy based on market mechanism (ARAS-M) [14] uses market based mechanism to allocate the resources to workloads based on QoS requirements specified by user. In this autonomic system, genetic algorithm (GA) is used to attain the equilibrium state by adjusting price automatically. This system fulfills the demand of every workload along with their QoS requirements but not effective to reduce SLA violations at runtime. Detecting SLA Violation infrastructure (DeSVi) [15] uses resource monitoring mechanism to prevent the violation of SLA. DeSVi allocates the resources to the workloads in virtual environment and resources are monitored by mapping user defined SLA with low-level resource metrics. DeSVi is not defined the limit of SLA deviation. Coordinated self-configuration of virtual machines (CoTuner) [16] uses model-free hybrid reinforcement learning technique to enable coordination among applications and virtual resources. CoTuner is working based on knowledge guided exploration policies to design a methodology for autonomic configuration of resources in case of fluctuation of workloads. Automated Resource allocation and cOnfiguration of MApreduce (AROMA) [17] allocates resources to workloads based on QoS requirements specified by cloud user. AROMA enables auto configuration of Hadoop jobs to compare the value of resource utilization with already executed workloads. High throughput cluster (HTC) computing system [18] is an extension of rock clusters to extend the local cluster to remote resources of cloud transparently and securely. HTC is working based on dynamic provisioning mechanism i.e. job scheduling policy in which database is updated regularly when new node is added or removed.

2.1.3 Self-optimizing

In cloud computing, self-optimizing is a capability of a system to improve the performance. Cloud auto scaling (CAS) [19] schedules activities of VM instance startup 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. CAS did not consider heterogeneous cloud workloads with different QoS parameters. Autonomic workload manager (AWM) [20] uses distributed provisioning and scaling decision making system (DPSMS) 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. AWM is not able to determine the cost of execution of workloads. Autonomic management framework (AMF) [21] uses an autonomic mechanism of performance and power management theoretically. AMF executes all the workloads on adequate resources with minimum execution time and energy but not consider cost. Bayesian network based decision support system (BN-DSS) [22] provides autonomic scaling of utility computing resources. BN-DSS system studies the historical behavior of autonomic system and predicts the performance, applicability and feasibility based on this historical data and negotiates the SLA. Mehdi et al. [23] 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 did not check the variation of resource contention along with number of workloads. EARTH [6] using fuzzy logic, which schedules the resources automatically to execute homogenous workloads. SOCCER [36] 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. SLA-aware autonomic resource management (STAR) technique [37] mainly focuses on SLA violation rate and also analyzed the impact of QoS parameters on SLA violation rate. Jose and Luıs [35] proposed a partial utility-driven resource scheduling (PURS) technique for elastic SLA and pricing negotiation which permits providers exchanging resources between VMs in expressive and economically effective ways. Further, a comprehensive cost method is defined by including partial utility given by customers to a definite level of degradation, when VMs are assigned in overcommitted situations. In this technique, revenue per resource allocation and execution time is improved.

2.1.4 *Self-protecting*

In cloud computing, self-protecting is a capability of a system to protect against the intrusions and threats. Secure autonomic technique (SAT) [24] is proposed to manage computing services and applications in secure environment. This technique protects the cloud environment with high accuracy of attack detection. Rainbow architecture based architecture based self-protection (ABSP) [25] is an autonomic technique in which security threats are detected at runtime through the use of patterns. ABSP reduces the security breaching and improves the depth of defense. Detection rate of attacks in ABSP is not as required. Self-healing and self-protection environment (SHAPE) [26] is an autonomic system to recover from various faults (hardware, software, and network faults) and protect from security attacks (distributed denial of service (DDoS), remote to local (R2L), user to root (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.

2.2 Comparisons of existing autonomic techniques with proposed approach [CHOPPER] based on properties of self-management

Based on existing literature [1,7–26,28,38–42] in QoS-aware autonomic cloud computing, we have classified the four properties of self-management (self-healing, self-configuring, self-protecting and self-optimization). Implementation of all the properties of self-management together is a challenging task and based on requirements and goals of an autonomic system, only some of the properties can be considered.

2.2.1 *Self-healing*

In cloud computing, self-healing improves the performance through *fault-tolerance* by reducing or avoiding the impact of failures on execution [1]. Failures occur in cloud due to following reasons: (1) unexpected changes of configuration of execution environment, (2) unavailability of resources, (3) overloading of resources, (4) shortage of memory, and (5) network failures. Existing autonomic systems are using techniques (check-pointing, failure forecasting and replication) to handle the above failures. *Check-pointing* technique is used to transfer the failed workload or tasks to the other available resources to start the execution from point of failure. *Failure forecasting* technique can be used to predict the requirement of resources in future in order to avoid failure of execution. In *replication* technique, workload is executed on more than one resource to increase the chances of successful execution.

2.2.2 *Self-configuring*

Self-configuring in cloud based autonomic systems is *installation* of missed or outdated components based on the alert generated by system without human intervention [1]. Some components may be *reinstalled* in changing conditions.

2.2.3 *Self-optimizing*

In cloud computing, self-optimizing is a capability of a system to improve the performance [1]. Dynamic scheduling techniques are being used in cloud to map the tasks or workloads on appropriate resources. *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 changed environment.

2.2.4 *Self-protecting*

To maintain the security and integrity of a system, it is required to detect and protect autonomic system from malicious attack [1]. To achieve this property of self-management, secure *scheduling policies* should be provided on both sides (provider side and user side). Security policies should be required in which system should be shut down before strong attack is about to happen. In *trust management* systems, malicious attackers can be detected through behavioral auditing. In *intrusion detection* technique, attacks are continually monitored and analyzed by the system to avoid future attacks. Comparisons of existing autonomic techniques with proposed approach [CHOPPER] based on properties of self-management is described in Tables 1, 2 and 3. In existing research work, Toulouse University Network (TUNe) and Java agent development environment (JADE) have been considered as autonomic manager [7,28,38–42].

2.3 Comparisons of existing autonomic techniques with proposed approach [CHOPPER] based on QoS parameters

Cloud based systems consider different QoS parameters to design a successful system. From literature [1,8–26], we have identified ten types of QoS parameters (fault detection rate, availability, reliability, security, cost, execution time, energy, SLA violation, resource contention and resource utilization) for use in autonomic cloud computing systems. *Fault Detection Rate* is the ratio of number of faults detected to the total number of faults existing. Faults may be software or hardware. *Availability* is an ability of a system to ensure the data is available with desired level of performance in normal as well as in fatal situations excluding scheduled downtime. *Reliability* is a capability of a system to perform consistently according to its predefined objectives. *Security* is ability to protect the data stored on cloud by using data encryption and access controls. *Energy* is amount of energy consumed by a resource to finish the execution of workload. *Execution time* is time required to execute the workload completely. *Cost* is an amount of the cost spent in one hour on the execution of workload. *Resource utilization* is a ratio of actual time spent by resource to execute workload to total uptime of resource for single resource. *SLA violation rate* is possibility of defilement of Service Level Agreement. When more than one workload shares same resource then *Resource Contention* may occur. It occurs due to following reasons: (i) when more than one workload executing on same resource, (ii) more number of workloads can create more resource contention and (iii) if number of provided resources are lesser than the number of required resources. By considering perspective of both cloud consumer and cloud provider, the comparison of existing autonomic techniques with proposed approach [CHOPPER] based on QoS parameters is described in Table 4.

All of the above research works have presented autonomic resource management techniques in cloud computing by focusing on different properties of self-management (self-healing, self-configuring, self-optimizing and self-protecting) with different QoS parameters. None of the existing works considered all the four properties of self-management simultaneously in a single cloud framework to test the different QoS parameters required in practical situations. Due to this the current autonomic resource management services become inefficient to respond in these situations.

2.4 Our contributions

In this paper, we have extended our previous research work self optimization of cloud computing energy-efficient resources (SOCCER) [36] by proposing QoS-aware autonomic resource management approach (CHOPPER). SOCCER clusters the heterogeneous cloud workloads and executes them with minimum energy consumption and focuses only on one aspect of self-optimization i.e. energy consumption. To consider other important aspects such as self-configuration, self-healing, self-protection and self-optimization, CHOPPER has been proposed. The major contributions of this paper are summarized as follows:

(i) CHOPPER offers self-configuration of cloud applications and resources by installation of missed or outdated components. Self-healing is provided by handling sudden failures, self-protection is offered against security attacks and self-optimization is as the resources are being used optimally. To reduce the human intervention and improve user satisfaction, CHOPPER automatically (through self-properties) manages QoS requirements of cloud users and schedules the provisioned cloud resources efficiently. Thus the cloud providers can achieve the SLAs and avoid SLA violations which offers better cloud service delivery. SOCCER focuses only on one aspect of self-optimization i.e. energy consumption but CHOPPER optimizes execution cost, time, energy efficiency and resource contention.

(ii) CHOPPER offers algorithms for three different phases (monitoring, analyses and plan as well as execution) based on autonomic properties. During execution of workloads, CHOPPER monitors the performance (QoS value) continuously, analyses the alert in case of performance degradation, plans an action to handle that alert and executes the plan to maintain the efficiency of CHOPPER.

(iii) CHOPPER offers workload classification into three categories based on deadline urgency: (a) normal (relaxed deadline), (b) good (average deadline) and (c) critical (urgent deadline) and investigates the impact of different workloads on different QoS parameters instead of first cum first serve execution of workloads.

(iv) CHOPPER increases security, energy efficiency, reliability and availability of cloud based services and reduces

Table 1 Comparison of existing autonomic techniques with proposed approach (CHOPPER) based on taxonomy of self-management

Technique	Self-healing	Self-configuration	Self-optimization	Self-protecting	Autonomic manager
CBR [12]	Fault prediction	Auto configuration/reinstallation	Adaptive scheduling	Scheduling policy	JADE
ASP [8]	NA	Auto configuration	Adaptive scheduling	NA	TUNe
COCCUS [13]	Check-pointing	Auto configuration/reinstallation	Dynamic scheduling	NA	JADE
CAS [19]	NA	Auto configuration	Dynamic scheduling	Trust management	TUNe
AWM [20]	Fault prediction	Auto configuration	Adaptive scheduling	Scheduling policy	JADE
AMF [21]	NA	Auto configuration/reinstallation	Adaptive scheduling/dynamic scheduling	NA	JADE
SH-SLA [9]	Fault prediction	NA	Dynamic scheduling	NA	TUNe
ARAS-M [14]	NA	NA	Adaptive scheduling/dynamic scheduling	NA	JADE
BN-DSS [22]	Fault prediction	Auto configuration	Adaptive scheduling	NA	JADE
SNOOZE [10]	Fault prediction/check-pointing	Auto configuration/reinstallation	Dynamic scheduling	Trust management	JADE
DeSVi [15]	NA	Auto configuration/reinstallation	Adaptive scheduling	NA	TUNe
AFRTC [11]	Fault prediction/check-pointing	NA	Dynamic scheduling	NA	JADE
CoTuner [16]	NA	Auto configuration	Dynamic scheduling	NA	JADE
AROMA [17]	NA	Auto configuration	Adaptive scheduling	Scheduling policy	TUNe
HTC [18]	Fault prediction	Auto configuration/reinstallation	Adaptive scheduling/dynamic scheduling	Scheduling policy	TUNe
SHAPE [26]	Fault prediction	NA	Dynamic scheduling	Intrusion detection	JADE
SAT [24]	NA	NA	NA	Trust management	TUNe
ABSP [25]	Fault prediction	NA	NA	Intrusion detection	JADE
ARCS [23]	Check-pointing	NA	Adaptive scheduling/dynamic scheduling		TUNe
PURS [35]	NA	Auto configuration/reinstallation	Adaptive scheduling	NA	TUNe
EARTH [6]	NA	NA	Adaptive scheduling/dynamic scheduling	NA	JADE
SOCCER [36]	NA	NA	Adaptive scheduling/dynamic scheduling	NA	JADE
STAR [37]	NA	NA	Adaptive scheduling/dynamic scheduling	NA	JADE
CHOPPER	Fault prediction/check-pointing	Auto configuration/reinstallation	Adaptive scheduling/dynamic scheduling	Scheduling policy/intrusion detection/trust management	JADE

Table 2 Comparison of existing autonomic techniques with proposed approach (CHOPPER) based on properties of self-management

Technique	Focus of study	Merits	Future possible extensions
CBR [12]	To reduce execution time and cost	CPU time and SLA violations are reduced	More QoS parameters like cost, time etc. can be considered
ASP [8]	To improve negotiation time for SLA	Reduced SLA violations and lease cost	Decision delay can be improved
COCCUS [13]	To reduce maintenance cost	Cost is reduced	Problem of starvation can be reduced
CAS [19]	To optimize resource utilization and cost	Average CPU time and Cost are reduced	Can be extended for network intensive applications
AWM [20]	To reduce monetary cost	Makespan is reduced	Penalty cost and compensation can be considered
AMF [21]	To reduce job's execution time	Execution time and network traffic are reduced	Stabilized API can be incorporated
SH-SLA [9]	To reduce SLA deviation	SLA violation is reduced	Heterogeneous workloads can be incorporated
ARAS-M [14]	To reduce energy consumption	Resource uptime is reduced	Failure prediction can be measured more accurately
BN-DSS [22]	Improves resource utilization	Resource contention is reduced	Problem of starvation can be reduced
SNOOZE [10]	To improve computational capacity	Time, scalability and cost are improved	Can be extended further to add different cloud providers
DeSVi [15]	To reduce resource consumption	Resource contention is reduced	Penalty cost and compensation can be considered for SLA violation
AFRTC [11]	To mprove resource utilization	Energy consumption is reduced	Average decision time can be reduced
CoTuner [16]	To reduce power consumption	No. of deadline missed, execution time, cost are reduced	For workloads, sensitivity of weight calculation can be considered
AROMA [17]	To improve resource utilization and execution time	Execution time is reduced	SLA violation can be reduced
HTC [18]	To reduce power consumption	Energy consumption and execution time are reduced	Cost can be reduced
SHAPE [26]	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
ARCS [23]	To reduce resource contention	Time of resource contention is reduced	Execution time and cost can be reduced
SAT [24]	To improve security	Improved attack detection rate	Self-optimization can be considered
ABSP [25]	To improve security	Reduced security breaching	Self-healing can be considered.
PURS [35]	To reduce SLA violation rate	Reduced SLA violation rate	Self-protecting can be considered
EARTH [6]	To reduce energy consumption of Homogenous Workloads	Improved energy efficiency for execution of homogenous workloads	Self-protecting, self-healing and self-configuration can be considered
SOCCER [36]	To reduce energy consumption of heterogeneous workloads	Improved energy efficiency for execution of heterogeneous workloads	Self-protecting, self-healing and self-configuration can be considered
STAR [37]	To reduce SLA violation rate and analyze its impact on QoS parameters	Reduced SLA violation rate and its impact on QoS parameters has been analyzed	Self-protecting, self-healing and self-configuration can be considered
CHOPPER	To design an intelligent cloud based QoS-aware autonomic resource management approach 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	CHOPPER manages resources automatically and offers 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) and it improves user satisfaction and increases reliability and availability of services.	CHOPPER can be extended by developing pluggable scheduler, in which resource scheduling policy can be changed easily based on the requirements

Table 3 Comparison of existing autonomic techniques with proposed approach (CHOPPER) based on main focus of study of different autonomic techniques

Technique	Self-healing	Self-configuration	Self-optimization	Self-protecting
CBR [12]		√		√
ASP [8]	√	√		
COCCUS [13]		√		
CAS [19]		√	√	√
AWM [20]			√	√
AMF [21]			√	
SH-SLA [9]	√			
ARAS-M [14]		√		
BN-DSS [22]			√	
SNOOZE [10]	√			
DeSVi [15]		√	√	
AFRTC [11]	√		√	
CoTuner [16]		√		
AROMA [17]		√	√	√
HTC [18]		√	√	√
SHAPE [26]	√			√
ARCS [23]			√	
SAT [24]				√
ABSP [25]				√
PURS [35]			√	
EARTH [6]			√	
SOCCER [36]			√	
STAR [37]			√	
CHOPPER	√	√	√	√

resource contention and SLA violation rate when implemented on a real cloud environment.

3 CHOPPER: QoS-aware autonomic resource management approach

This section discusses the architecture of configuring, healing, optimizing and protecting policy for efficient resource management (CHOPPER) which requires that QoS parameters must be described in the form of SLA. CHOPPER is the key mechanism that ensures that cloud providers can serve large amount of requests without violating SLA terms and dynamically manages the resources based on QoS requirements described by user and workload characteristics. Architecture of CHOPPER is shown in Fig. 1. QoS-aware autonomic resource management approach comprises of following units:

3.1 Cloud workload management portal

First of all cloud consumer tries to execute the workloads through the cloud workload management portal (CWMP)[1]. Web browser acts as an interface for consumer to interact with CHOPPER. After that, the task of cloud consumer's authorization and authentication is performed. After authentication, CHOPPER asks to submit the cloud consumer requirements (SLA) and authenticated cloud consumer fills it and submits the request for the availability of particular resource with proper specification for the execution of their workload. CHOPPER takes the information from the appropriate workload after analyzing the various workload details which cloud consumer demanded. For multi-tenancy, we have considered different cloud providers which are interacting with each other by using CWMP and update their new rules and policies of resources on cloud in the proposed approach as shown in Fig. 1.

3.2 Workload manager

The aim of Workload Manager is to look at different characteristics of a cloud workload to determine the feasibility of porting the application in the cloud. The different cloud workloads have different set of requirements and characteristics. This analysis also provides input to execution method. It comprises of three sub units: bulk of workloads, workload description and workload queue. All the workloads submitted by cloud consumer for execution is considered as bulk of

[1] http://hdl.handle.net/10266/2247

Table 4 Comparison of existing autonomic techniques with proposed approach (CHOPPER) based on QoS parameters

Technique	Cloud consumer						Cloud provider			
	Availability	Reliability	Security	Cost	Execution time	SLA violation rate	Energy consumption	Resource utilization	Fault detection Rate	Resource contention
CBR [12]			✓	✓	✓	✓		✓		
ASP [8]				✓		✓				
COCCUS [13]				✓	✓					
CAS [19]			✓	✓	✓					
AWM [20]			✓		✓			✓		
AMF [21]					✓		✓			
SH-SLA [9]	✓	✓				✓		✓		
ARAS-M [14]				✓						
BN-DSS [22]					✓					
SNOOZE [10]			✓	✓	✓			✓		
DeSVi [15]				✓	✓	✓				
AFRTC [11]	✓	✓								
CoTuner [16]								✓		
AROMA [17]			✓	✓	✓			✓		
HTC [18]			✓		✓					
SHAPE [26]			✓	✓	✓			✓		
SAT [24]			✓							
ABSP [25]			✓							
ARCS [23]										✓
PURS [35]						✓				
EARTH [6]							✓	✓		
SOCCER [36]							✓	✓		
STAR [37]						✓				
CHOPPER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

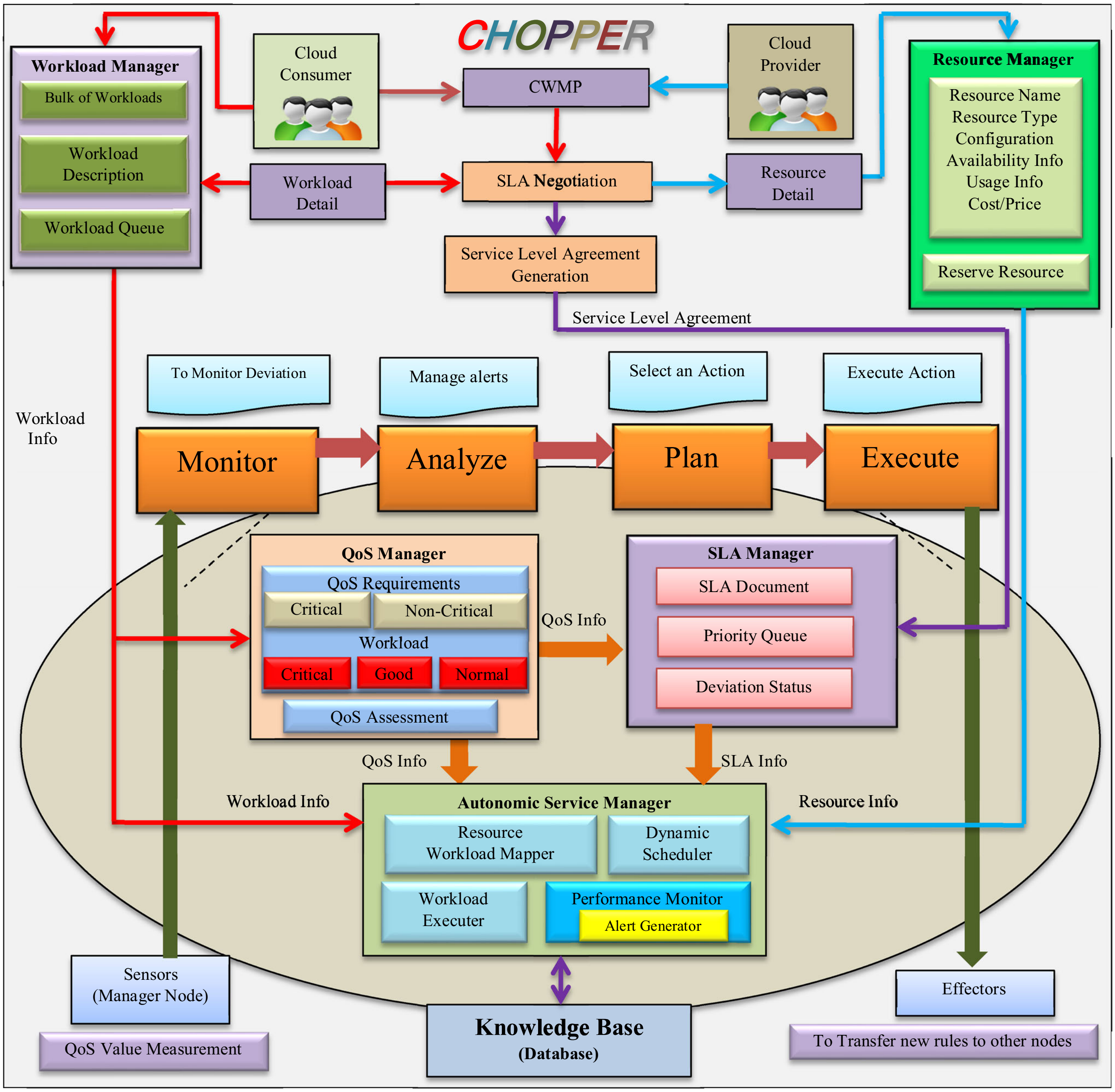


Fig. 1 Chopper architecture [AE]

workloads. In workload description, all the workloads should have their key QoS requirements, based on that, the workload is executed with some user defined constraints. Being able to calculate the execution time of workloads in advance is one of the fundamental assumptions because workload gets accommodated in the queue and during this period, its time can be statistically calculated on the basis of its size [like kilo lines of code (KLoC) or function point (FP)], where KLoC would be more appropriate or size can be taken as an input from cloud consumer. The types of workload that have been considered for this research work are: websites, technological 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 [4,5]. After analysis of workloads, they are classified on the basis of specific features in terms of security needs, network needs, variability of load, back-up services, network bandwidth needs, computing capacity and other QoS metrics. In workload queue**,** all the feasible cloud workloads are put into a workload queue for provisioning of resources

Table 5 Cloud workloads, workload type and their QoS requirements

Workload name	QoS requirements	Workload type
Websites	Reliable storage, High network bandwidth, High availability	Communication
Technological computing	Computing capacity	Compute
Endeavour software	Security, high availability, customer confidence level, correctness	Administration
Performance testing	Time, cost, energy, resource utilization and SLA violation rate	Compute
Online transaction processing	Security, high availability, internet accessibility, usability	Administration
E-commerce	Variable computing load, customizability	Storage
Central financial services	Security, high availability, changeability, integrity	Administration
Storage and backup services	Reliability, persistence	Storage
Productivity applications	Network bandwidth, latency, data backup, security	Administration
Software/project development and testing	User self-service rate, flexibility, creative group of infrastructure services, testing time	Administration
Graphics oriented	Network bandwidth, latency, data backup, visibility	Administration
Critical internet applications	High availability, serviceability, usability	Communication
Mobile computing services	High availability, reliability, portability	Communication

before actual execution. Finally, workload generates output in the form of workload information. We have identified the QoS requirements for every workload and their corresponding QoS metrics has been designed [4,5] is described in Table 5. Based on different QoS parameters of different workloads, only required metrics have been applied to test the performance of CHOPPER.

The assumptions of proposed approach are: (a) Multiple users can access the cloud based system simultaneously, (b) Workloads have different execution time and (c) Workloads have different deadlines.

3.3 Resource manager

The resource details include the number of CPUs used, size of memory, cost of resources, type of resources and number of resources. All the common resources are stored in resource pool and reserve pool contains some reserve resources. It contains the information about the available resources and reserved resources along with resource description (resource name, resource type, configuration, availability information, usage information and price of resource) as provided by cloud provider.

3.4 Service level agreement

SLA describes what you require from your consumers/service customers in order to provide the service specified. It needs assurance and support from both parties to provision and follow the contract in order to ensure SLA fulfillment. Based on the QoS requirements described by consumer, different set of requirements and characteristics of different workloads and resource detail provided by the provider, final SLA is signed after SLA negotiation. Finally, signed document is submitted to SLA manager. The interaction of cloud user and cloud provider to negotiate SLA as described in our previous research work [37].

3.5 QoS manager

It comprises of two sub units: QoS requirements and QoS assessment. Based on the key QoS requirements of a particular workload (workload information generated by workload manager), the QoS manager puts the workload into critical (urgent workloads) and non-critical queues (non-urgent workloads) through QoS assessment [2] as shown in Fig. 2. For QoS assessment, QoS Manager calculates the execution time of workload using (Eq. 1) and finds the approximate workload completion time (addition of waiting time and execution time) using (Eq. 3). If the completion time is less than the desired deadline then the workload will be executed immediately with the available resources and the resource(s) would be released back to resource manager for another execution otherwise calculate extra number of resources required and provide them from the reserved pool for current execution as shown in Fig. 2. The first state for every workload is submission, based on key QoS requirements of workload, the next state is decided either as critical (critical workload) or non-critical (good workload or normal workload).

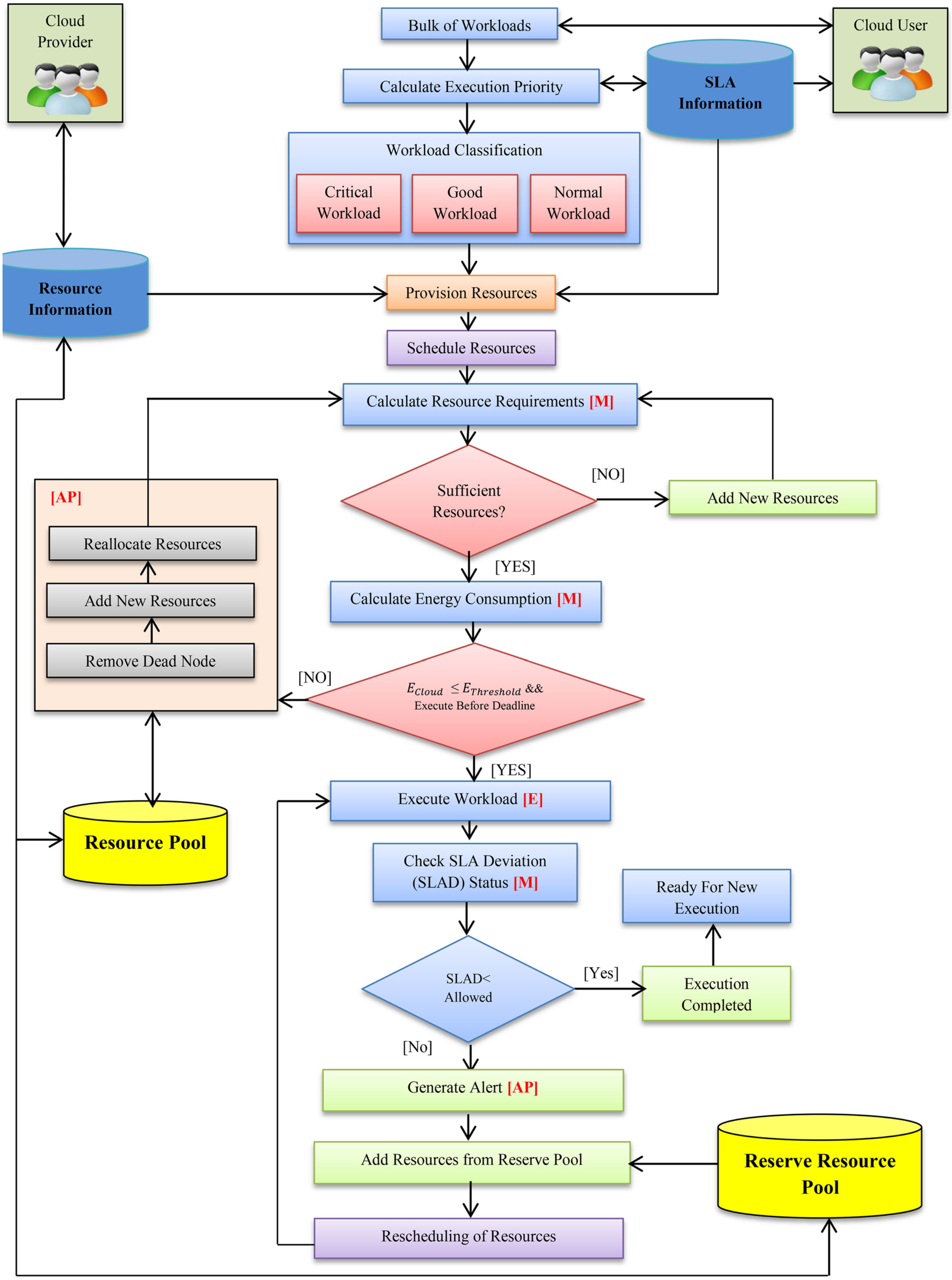


Fig. 2 Execution of workloads in CHOPPER

After non-critical state, if there is no other workload before that then it is executed directly otherwise put current workload into non-critical queue for waiting. After successful execution of workload, the workload is completed. On the other hand, all the QoS oriented workloads are put into critical queue and sorted based on their priority decided by QoS manager based on SLA information and then scheduled for execution. If there is no obstacle (urgency, more resource requirement etc.) then execute directly with available resources otherwise put into under scheduling state to fulfill the user requirements. If all the conditions will meet in the given budget, resource and time constraints then it will execute otherwise it will not be executed. For instance, when a workload requires low amount of resources, it will assign resources with lower capability, so that new requests can be served in an efficient manner.

3.6 QoS based metrics

The following metrics are selected from our previous work [4–6,32,36,37] to calculate the execution time, execution cost, energy consumption and waiting time.

3.6.1 Execution time

It is a ratio of difference between workload finish time (WF_i) and workload execution start time (WE_i) to number of workloads. We have used following formula to calculate execution time (ET) (Eq. 1).

$$ET_i = \sum_{i=1}^{n} \left(\frac{WF_i - WE_i}{n} \right) \tag{1}$$

Where n is the number of workloads to be executed.

3.6.2 Waiting time

It is a ratio of difference between workload execution start time (WE_i) and workload submission time (WS_i) to number of workloads. We have used following formula to calculate waiting time (Eq. 2).

$$Waiting\ Time_i = \sum_{i=1}^{n} \left(\frac{WE_i - WS_i}{n} \right) \tag{2}$$

where n is the number of workloads. Completion time (Eq. 3) of workload is addition of waiting time and execution time.

$$Completion\ Time = ET_i + Waiting\ Time_i \tag{3}$$

3.6.3 Energy consumption

The energy model is devised on the basis that resource utilization has a linear relationship with energy consumption (E_{cloud}) as described in SOCCER [36]. Energy model used to calculate energy consumption of resources is described in our previous research work [36]. Only those resources are provisioned which have $E_{cloud} < E_{threshold}$, where $E_{threshold}$ is a threshold value of energy consumption.

3.6.4 Average cost

It is an addition of resource cost and penalty cost. CHOPPER defined the different levels of penalty rate based on QoS requirements. Delay time is difference of deadline and time when workload is actually completed [37]. We have used following formula to calculate average cost (C) (Eq. 4).

$$C = Resource\ Cost + Penalty\ Cost \tag{4}$$

$$Resource\ Cost = ET_i \times Price \tag{5}$$

$$Penalty\ Cost = \sum_{i=1}^{C} (PC_i) \tag{6}$$

$$Delay\ Time = Expected\ Completion\ Time - Actual\ Completion\ Time \tag{7}$$

$$PC = \begin{cases} Penalty_{\min imum} & if\ Expected\ Completion\ Time \geq Actual\ Completion\ Time \\ Penalty_{\min imum} + [Penalty\ Rate \times |DelayTime| & if\ Expected\ Completion\ Time \geq Actual\ Completion\ Time \end{cases} \tag{8}$$

Where $c \in PC$, PC is set of penalty cost with different levels specified in CHOPPER. The *Complexity* (Eq. 9) of resource scheduling policy is influenced by number of Change Points (CP) i.e. rescheduling and requested resources (r) of *Workload* being scheduled. Here,

$$CP = (S_t + T_e + T_s + R_t) \tag{9}$$

where, S_t = start time of all workloads, T_s = suspend time of all workloads, T_e = end time of all workloads and R_t = resume time of all workloads. The mapping is done with the objective of minimum cost, time and energy simultaneously. Consider lesser pre-emption as its objective. The complexity of the algorithms mainly in this research work depends on two important objectives:

- Minimize the rejection rate of the incoming requests.
- Minimize reshuffle cost (avoid rescheduling of already accommodated leases as much as possible).

3.7 SLA manager

Based on SLA information (signed service level agreement), SLA document will be prepared which contains information about SLA violation (maximum deviation, minimum deviation and penalty rate in case of SLA violation) and accordingly urgent cloud workloads would be placed in priority queue for earlier execution as shown in Fig. 2. Deviation status is used to measure the deviation of QoS from predictable values with their possible resolution. In case of urgent workloads, if the deviation is more than the allowed then penalty will be imposed (it will allocate the reserve resources to the particular workload for compensation) as shown in Table 7. For example: In SLA, both the parties (cloud provider and cloud consumer) should have specified the possible deviations to achieve appropriate quality attributes. For example: if we consider *availability* as a *quality attribute* and it should be 95%, this implies the system should be available for 22.8 h per day with maximum deviation of 1.2 h per day (5%). In case of system performance, if the desired deadline is 9 ms with deviation (10%) of 1ms then maximum response time should be 10 ms for a particular task without violation of agreement. Cloud provider's SLA will give an indication of how much actual availability of service the provider views as adequate, and to what amount it is agreeable to require its own financial resources to compensate for unexpected outages [2]. If there is violation of SLA (misses the deadline), then penalty delay cost is imposed or compensation is given to consumer as described in Sect. 4.2.1. Penalty cost is equivalent to how much the service provider has to give concession to users for SLA violation (Eq. 8). It is dependent on the penalty rate and penalty delay time period. In CHOPPER, the effect of inaccuracy can be reduced by two ways: (i) consider the penalty compensation clause in SLAs with provider and impose SLA violation and (ii) add some slack time during scheduling for avoiding risk. This research work considers 5% as minimum deviation and 15% as maximum deviation. In this research work, workloads are classified into three categories based on deadline urgency: (i) normal (relaxed deadline), (ii) good (average deadline) and (iii) critical (urgent deadline) as discussed in Sect. 4.2.1.

We have selected the "web services agreement specification (WS-Agreement)" standard [34] for management of SLA in this research work. WS-agreement protocol is used for establishing agreement between two parties, such as between a cloud provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties. The specification consists of a schema for specifying an agreement, a schema for specifying an agreement template, and a set of classes for managing agreement life-cycle, including creation, expiration, and monitoring of agreement states through activity diagram [33]. *SLA manager* manages the whole service of CHOPPER in a controlled manner and uses minimum number of resources to execute the workloads within specified budget and deadline along with minimum energy consumption.

3.8 Autonomic service manager

Based on information of SLA, QoS, workload and resource, the resource workload mapper maps the workloads to the appropriate resources by taking care of both SLA and QoS. Dynamic scheduler schedules the workloads after mapping of the workloads with available resources based on the policy (cost, time, cost-time and bargaining based [5]) defined by user and generates the workload schedule based on the workload details specified by the user and billing for that execution. Energy is also calculated as discussed in Sect. 3.6.3 and compared with threshold energy value at different value of resources. The workload is dispatched only, if the workload is executed within described budget and deadline and actual energy consumption is less than the threshold energy value. After verification of every critical parameter the workloads are dispatched for execution. After payment, the workload executer will execute the workloads. CHOPPER mainly focuses on the properties of self-management i.e. self-healing, self-configuring, self-protecting and self-optimizing.

Self-healing Self-healing in CHOPPER aims to make all the necessary changes to recover from the faults 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 any unhandled exception 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 and network breakage in case of distributed networks.

Self-protecting The main aim of self-protecting in CHOPPER is to protect the system from malicious 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 DoS, R2L, U2R and probing. In denial of service (DoS) 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 towards broadcast IP address), local area network denial (LAND) (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 destroy the network by executing their commands. It includes attacks like internet message access protocol (IMAP), Guess password and SPY. In user to root (U2R), attackers get root access of the system to destroy the network. It includes attacks like buffer overflow and rootkits. In Probing, attackers use programming language to steal the private information. It includes attacks like port sweep and Network MAPper (NMAP).

Self-configuring The main aim of self-configuring in CHOPPER 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 takes care of cost which includes cost of resource and penalty cost in case of SLA violation.

Self-optimizing Self-optimizing aspect of CHOPPER maps the tasks or workloads to the appropriate resources using dynamic scheduling techniques. 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 changed environment. Self-optimizing is effected by various QoS parameters such as execution time, execution cost, resource utilization, availability of service, reliability of service, energy efficiency and resource contention. CHOPPER combines self-healing, self-configuring self-protecting and self-optimizing approach for complex cloud based distributed systems. Autonomic elements (AEs) are mainly responsible for autonomic management of resources. AE consists of sensors, monitor, analyzer, planner, executor and effector as shown in Fig. 1. All the AEs interact with each other periodically for updated information regarding system performance. Based on interaction, AEs finish a mandatory subtask to maintain the performance of CHOPPER. Autonomic unit (AU) is a set of AEs working together for the achievement of a particular task. AU has one manager node (CHOPPER-Manager) and more than one processing nodes (CHOPPER-Client), all the processing nodes report to the manger node and only manager node of one AU can interact with manager node of another AU. Mathematically, we can describe a relation as:

$$CHOPPER = \{AU_1, AU_2, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots AU_m\} while AU = \{AE_1, AE_2, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots AE_m\}$$

QoS-aware autonomic resource management approach executes the workloads as shown in Fig. 2. Firstly, bulk of workloads submitted are placed into workload queue based on feasibility and their QoS requirements. Based on SLA information, resource information, workload information and QoS information resources are provisioned by using Q-aware resource provisioning technique [4]. After provisioning of resources, actual resource scheduling is done based on QRSF resource scheduling technique [5]. After scheduling of resources, actual execution of workloads is started. During execution of workloads, performance is monitored continuously using a sub unit *performance monitor* to maintain the efficiency of CHOPPER that generates alert in case of performance degradation. Alerts can be generated in two conditions generally: (i) if there are insufficient resources available to execute workload (*Action:* Reallocates resources) and (ii) if the SLA deviation is more than allowed (*Action*: Negotiate SLA) as shown in Fig. 2. Working of sub units is described in Fig. 2 as: Monitor [M], analyze and plan [AP] and executor [E]. Other possible alerts and corresponding actions are discussed below in next sections. Same action is performed twice, if AE fails to correct it then system is treated as down. Java agent development environment (JADE) is used to establish the communication among AEs and exchanging information for updates and all the updated information is stored in centralized database (Knowledge Base) for future usage and backup of corresponding updates is also maintained in case of failure of database. Java development kit public key infrastructure (JDK-PKI) [26] plugin is used to provide security that ensures the privacy of communication among AEs. Working of AE of CHOPPER is based on IBM's autonomic model [6,27] that considers four steps of autonomic system: (i) monitor, (ii) analyze, (iii) plan and (iv) execute. Autonomic system based CHOPPER comprises of following components:

3.8.1 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. If the value of energy consumption is less than threshold value 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] and transfers updated information to manager node.

3.8.2 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 knowledge base. 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 transfer this information to next module for further analysis [ALGORITHM 1: Monitoring Uni (MU)] is used to monitor the performance of management of resources by considering four self-management properties as shown in Fig. 3. 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 $\leq$ B_E] or [E_{Cloud} $\leq$ $E_{Threshold}$]) will be false then alert will be generated otherwise schedule resources for execution. Where D_t(deadline time) is calculated based on desired deadline using (Eqs. 10, 11), and B_E is maximum budget allocated for execution [4,5]. Equation 10 is used to calculate $deadlinetime(Dt_i)$.

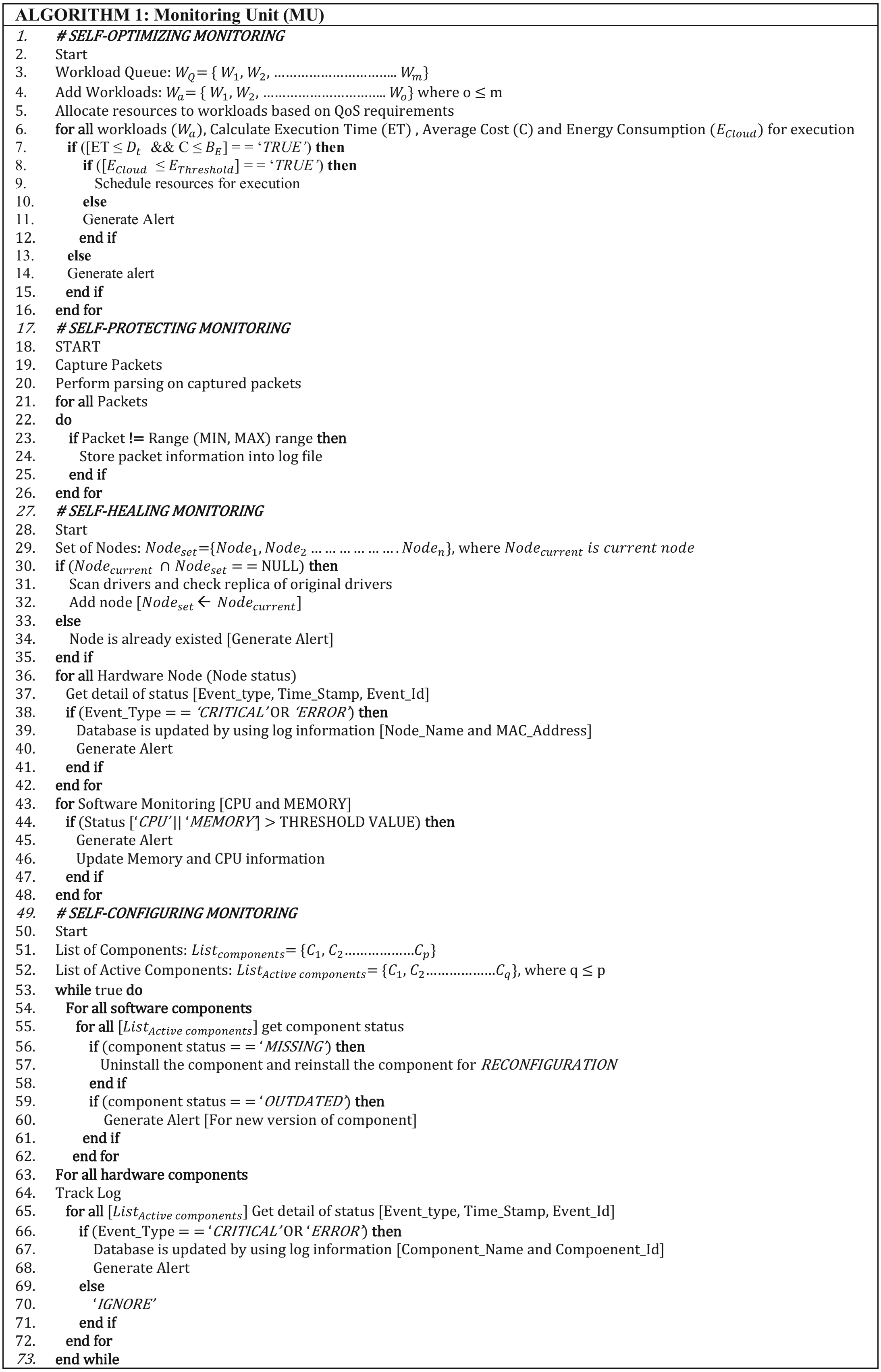
$$Dt_i = \sum_{i=1}^{n} (Wd_i - Ct_i) \tag{10}$$

Where Wd_i is workload deadline and Ct_i is current time. Deadline urgency (Du_i) is used to calculate the final priority of workload (Eq. 11).

$$\mathrm{D}u_i = \sum_{i=1}^{n} \left(\frac{Dt_i}{ET_i} - 1 \right) \tag{11}$$

Where Dt_i is deadline time and ET_i is execution time calculated using (Eq. 1). 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 CHOPPER. CHOPPER 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 existing database. New anomalies are captured by security agent and information about anomalies is stored in database (knowledge base). CHOPPER protects the system from various attacks as discussed earlier such as DoS [Smurf, LAND, SYN Flood and Teardrop], R2L [SPY, Guess password, IMAP], U2R [rootkits, buffer overflow] and probing [ports sweep and NMAP]. SNORT anomaly detector [26] is used to protect the system from attacks. In our experimental work, we have integrated SNORT with CHOPPER (Fig. 6). We have used detection engine to detect the attacks and maintain the log about attack. 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)]. State vector machine [26] is used in CHOPPER to make a network profile for attack detection. State Vector Machine is designed based on training data 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, hardware, network, 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 is already existing in the system then it will generate alert. CHOPPER performs the hardening [26] 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 replace the existing drivers. If any alert is generated after hardening of driver then original driver replace 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 CHOPPER to resolve hardware failures and generate alert in case of any internal error and store the information regarding alert into database. CHOPPER 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. CHOPPER fixes some threshold value usage for both CPU and memory. 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



ALGORITHM 1: Monitoring Unit (MU)

```
# SELF-OPTIMIZING MONITORING
Start
Workload Queue: $W_Q$= { $W_1$, $W_2$, ............................... $W_m$}
Add Workloads: $W_a$= { $W_1$, $W_2$, ............................... $W_o$} where o ≤ m
Allocate resources to workloads based on QoS requirements
for all workloads ($W_a$), Calculate Execution Time (ET) , Average Cost (C) and Energy Consumption ($E_{Cloud}$) for execution
  if ([ET ≤ $D_t$  && C ≤ $B_E$] = = 'TRUE') then
    if ([$E_{Cloud}$ ≤ $E_{Threshold}$] = = 'TRUE') then
      Schedule resources for execution
    else
    Generate Alert
    end if
  else
  Generate alert
  end if
end for
# SELF-PROTECTING MONITORING
START
Capture Packets
Perform parsing on captured packets
for all Packets
do
  if Packet != Range (MIN, MAX) range then
    Store packet information into log file
  end if
end for
# SELF-HEALING MONITORING
Start
Set of Nodes: $Node_{set}$={$Node_1$, $Node_2$ … … … … … …. $Node_n$}, where $Node_{current}$ $is\ current\ node$
if ($Node_{current}$ ∩ $Node_{set}$ = = NULL) then
  Scan drivers and check replica of original drivers
  Add node [$Node_{set}$ ← $Node_{current}$]
else
  Node is already existed [Generate Alert]
end if
for all Hardware Node (Node status)
  Get detail of status [Event_type, Time_Stamp, Event_Id]
  if (Event_Type = = 'CRITICAL' OR 'ERROR') then
    Database is updated by using log information [Node_Name and MAC_Address]
    Generate Alert
  end if
end for
for Software Monitoring [CPU and MEMORY]
  if (Status ['CPU'|| 'MEMORY'] > THRESHOLD VALUE) then
    Generate Alert
    Update Memory and CPU information
  end if
end for
# SELF-CONFIGURING MONITORING
Start
List of Components: $List_{components}$= {$C_1$, $C_2$.................$C_p$}
List of Active Components: $List_{Active\ components}$= {$C_1$, $C_2$.................$C_q$}, where q ≤ p
while true do
  For all software components
    for all [$List_{Active\ components}$] get component status
      if (component status = = 'MISSING') then
        Uninstall the component and reinstall the component for RECONFIGURATION
      end if
      if (component status = = 'OUTDATED') then
        Generate Alert [For new version of component]
      end if
    end for
For all hardware components
Track Log
  for all [$List_{Active\ components}$] Get detail of status [Event_type, Time_Stamp, Event_Id]
    if (Event_Type = = 'CRITICAL' OR 'ERROR') then
      Database is updated by using log information [Component_Name and Compoenent_Id]
      Generate Alert
    else
      'IGNORE'
    end if
  end for
end while
```

Fig. 3 Algorithm for monitoring in CHOPPER

network. CHOPPER 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 used at different processing nodes, status of active component is retrieved by software component agent. In CHOPPER, two types of status are defined in database: 'MISSING' or 'OUTDATED'. If software component agent generates 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, CHOPPER 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.

3.8.3 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. [ALGORITHM 2: Analyzing and Planning Unit (APU)] is used to analyze the performance of management of resources by considering four self-management properties as shown in Fig. 4. 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] as described in Fig. 2. For *self-protecting*, the analyzing unit starts analyzing the log information of attacks after alert is generated by security agent to generate signature. CHOPPER performs following function to generate signature:

- Collect all the new alerts generated by autonomic element (AE)
- Use Java utility to perform parsing to get URL, port and payload detail
- Categorize data based on URL, port and payload
- To find largest common substring apply longest common subsequence (LCS)
- 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 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 taking more time to execute or usage of CPU or memory are 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 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.

3.8.4 Executor [E]

Executor implements the plan after analyzing completely. [ALGORITHM 3: Executing Unit (EU)] is used to execute the resource and analyze the execution performance by considering four self-management properties as shown in Fig. 5.

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

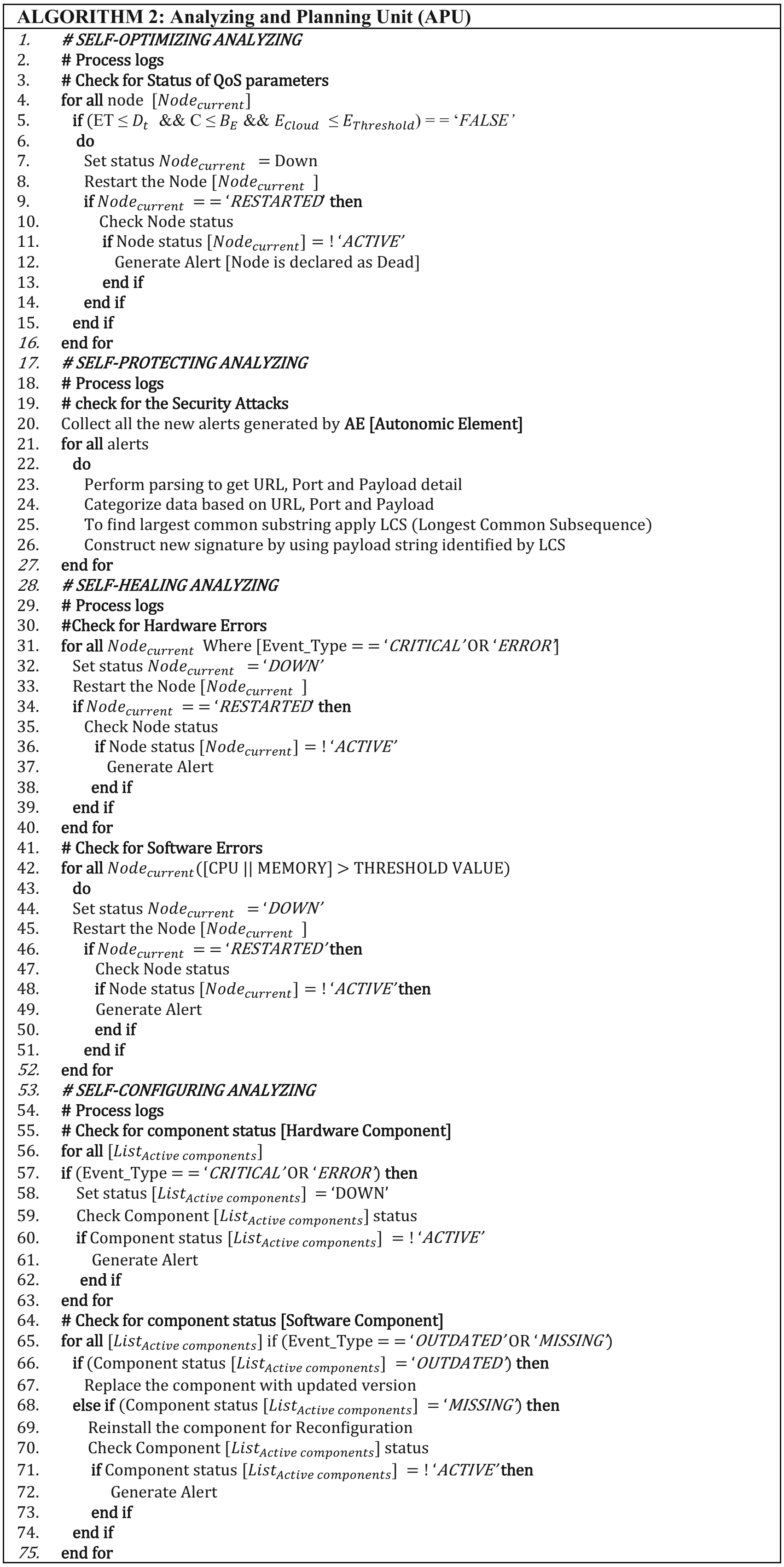
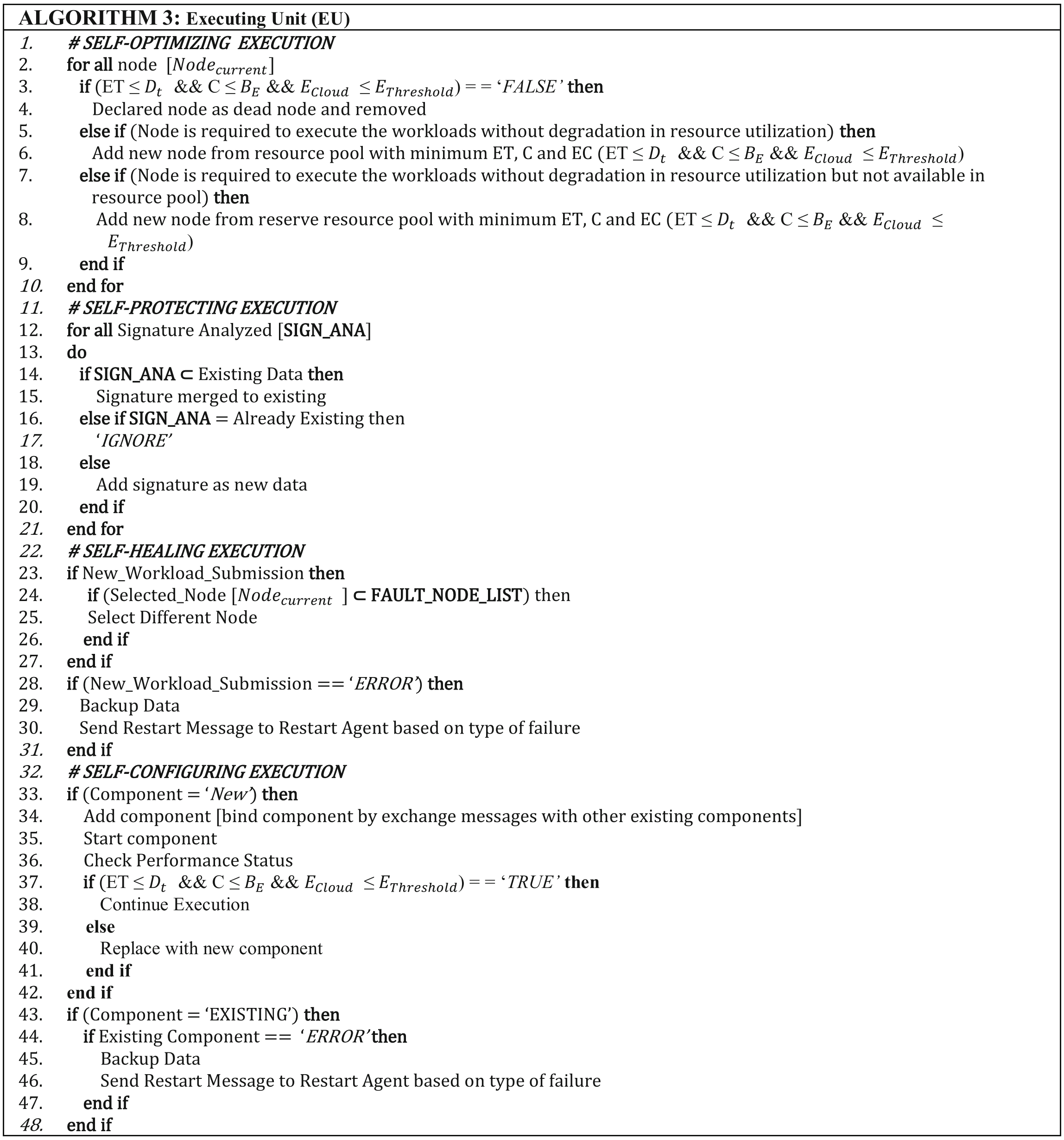


Fig. 4 Algorithm for analyzing and planning in CHOPPER

ALGORITHM 2: Analyzing and Planning Unit (APU)

```
# SELF-OPTIMIZING ANALYZING
# Process logs
# Check for Status of QoS parameters
for all node [$Node_{current}$]
  if (ET ≤ $D_t$ && C ≤ $B_E$ && $E_{Cloud}$ ≤ $E_{Threshold}$) = = 'FALSE'
   do
    Set status $Node_{current}$ = Down
    Restart the Node [$Node_{current}$ ]
    if $Node_{current}$ = = 'RESTARTED' then
       Check Node status
       if Node status [$Node_{current}$] = ! 'ACTIVE'
          Generate Alert [Node is declared as Dead]
       end if
    end if
  end if
end for
# SELF-PROTECTING ANALYZING
# Process logs
# check for the Security Attacks
Collect all the new alerts generated by AE [Autonomic Element]
for all alerts
  do
    Perform parsing to get URL, Port and Payload detail
    Categorize data based on URL, Port and Payload
    To find largest common substring apply LCS (Longest Common Subsequence)
    Construct new signature by using payload string identified by LCS
end for
# SELF-HEALING ANALYZING
# Process logs
#Check for Hardware Errors
for all $Node_{current}$ Where [Event_Type = = 'CRITICAL' OR 'ERROR']
  Set status $Node_{current}$ = 'DOWN'
  Restart the Node [$Node_{current}$ ]
  if $Node_{current}$ = = 'RESTARTED' then
    Check Node status
      if Node status [$Node_{current}$] = ! 'ACTIVE'
        Generate Alert
     end if
  end if
end for
# Check for Software Errors
for all $Node_{current}$([CPU || MEMORY] > THRESHOLD VALUE)
  do
  Set status $Node_{current}$ = 'DOWN'
  Restart the Node [$Node_{current}$ ]
    if $Node_{current}$ = = 'RESTARTED' then
      Check Node status
      if Node status [$Node_{current}$] = ! 'ACTIVE' then
      Generate Alert
      end if
    end if
end for
# SELF-CONFIGURING ANALYZING
# Process logs
# Check for component status [Hardware Component]
for all [$List_{Active\ components}$]
if (Event_Type = = 'CRITICAL' OR 'ERROR') then
   Set status [$List_{Active\ components}$] = 'DOWN'
   Check Component [$List_{Active\ components}$] status
   if Component status [$List_{Active\ components}$] = ! 'ACTIVE'
      Generate Alert
   end if
end for
# Check for component status [Software Component]
for all [$List_{Active\ components}$] if (Event_Type = = 'OUTDATED' OR 'MISSING')
  if (Component status [$List_{Active\ components}$] = 'OUTDATED') then
    Replace the component with updated version
  else if (Component status [$List_{Active\ components}$] = 'MISSING') then
     Reinstall the component for Reconfiguration
     Check Component [$List_{Active\ components}$] status
      if Component status [$List_{Active\ components}$] = ! 'ACTIVE' then
          Generate Alert
      end if
  end if
end for
```



```
ALGORITHM 3: Executing Unit (EU)
# SELF-OPTIMIZING EXECUTION
for all node [$Node_{current}$]
  if (ET ≤ $D_t$ && C ≤ $B_E$ && $E_{Cloud}$ ≤ $E_{Threshold}$) = = 'FALSE' then
    Declared node as dead node and removed
  else if (Node is required to execute the workloads without degradation in resource utilization) then
    Add new node from resource pool with minimum ET, C and EC (ET ≤ $D_t$ && C ≤ $B_E$ && $E_{Cloud}$ ≤ $E_{Threshold}$)
  else if (Node is required to execute the workloads without degradation in resource utilization but not available in
    resource pool) then
     Add new node from reserve resource pool with minimum ET, C and EC (ET ≤ $D_t$ && C ≤ $B_E$ && $E_{Cloud}$ ≤
       $E_{Threshold}$)
  end if
end for
# SELF-PROTECTING EXECUTION
for all Signature Analyzed [SIGN_ANA]
do
  if SIGN_ANA ⊂ Existing Data then
     Signature merged to existing
  else if SIGN_ANA = Already Existing then
     'IGNORE'
  else
     Add signature as new data
  end if
end for
# SELF-HEALING EXECUTION
if New_Workload_Submission then
    if (Selected_Node [$Node_{current}$ ] ⊂ FAULT_NODE_LIST) then
    Select Different Node
   end if
end if
if (New_Workload_Submission == 'ERROR') then
  Backup Data
  Send Restart Message to Restart Agent based on type of failure
end if
# SELF-CONFIGURING EXECUTION
if (Component = 'New') then
   Add component [bind component by exchange messages with other existing components]
   Start component
   Check Performance Status
   if (ET ≤ $D_t$ && C ≤ $B_E$ && $E_{Cloud}$ ≤ $E_{Threshold}$) = = 'TRUE' then
      Continue Execution
    else
      Replace with new component
    end if
end if
if (Component = 'EXISTING') then
   if Existing Component == 'ERROR' then
      Backup Data
      Send Restart Message to Restart Agent based on type of failure
   end if
end if
```

Fig. 5 Algorithm for execution in CHOPPER

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 as shown in Fig. 2. If still issue is not resolved then generate alert. Figure 2 also describes how CHOPPER reacts when a workload fails during its execution and self-healing capability of CHOPPER to re-negotiate the SLAs based on availability of resources (reserve resources can be used in case of unavailability of resources in resource pool with new SLA). Resources are adding in resource pools through resource provisioning (Q-aware). 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. If still issue is not resolved then generate alert. For *self-protecting*, SNORT 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 (knowledge base) 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. If still issue is not resolved then generate alert.

3.8.5 Effector

Effector is acting as an interface between AUs and AEs to exchange updated information and it is used to transfer the new policies, rules and alerts to other nodes with updated information.

4 Implementation and experimental results

We modeled and simulated a cloud environment using CloudSim [31]. We simulated computing nodes that resembles configuration of resources shown in Table 6. The workload is modeled as processing of images to convert from one format to another (e.g., converting from JPEG to PNG format). Microsoft Visual Studio 2010 is an integrated development environment from microsoft [32]. JADE is used to establish the communication among devices and exchanging information for updates and all the updated information is stored in centralized database for future usage and backup of corresponding updates is also maintained in case of failure of database. SNORT [26] is the most commonly used signature-based detector that runs over IP networks for analyzing real time traffic for detection of misuse. SNORT also provides the option to make it work as anomaly detection IDS by using the preprocessor component. CHOPPER is using the SNORT anomaly detector version to self-protect the system from security attacks. SNORT has been optimized to be integrated with CHOPPER. For Self-protection, "analysis signatures" generated by analyzer are further refined and finalized to be used as a signature by SNORT. For this, analyzed signatures are compared with existing signatures in the SNORT database. CloudSim [31] has been installed along with its requirements on all the nodes which are participating to provide cloud service. Nodes in this system can be added or removed based on the requirement. We have verified CHOPPER in real cloud environment. The integration of multiple environments used to conduct experiments is shown in Fig. 6.

Energy consumption is measured in Kilo Watt Hour (kWh) using *Joule Meter*. CHOPPER is installed on main server and tested on virtual cloud environment that has been established at *CLOUDS Lab, School of Computing and Information Systems, The University of Melbourne, Australia*. We installed different number of virtual machines on three different servers, and deployed CHOPPER to measure the variations. In this experimental setup, three different cloud platforms are used: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). At *software level*, Microsoft Visual Studio 2010 is used to develop Cloud Workload Management Portal (CWMP) to provide user interface in which user can access service from any geographical location.

In this research work, CloudSim toolkit is used [31]. Both the behavior and system modeling of Cloud system components (VMs, datacenters and RP policies) is supported by this toolkit. It is used to implement the common resource scheduling techniques through little effort and can

Fig. 6 Cloud testbed

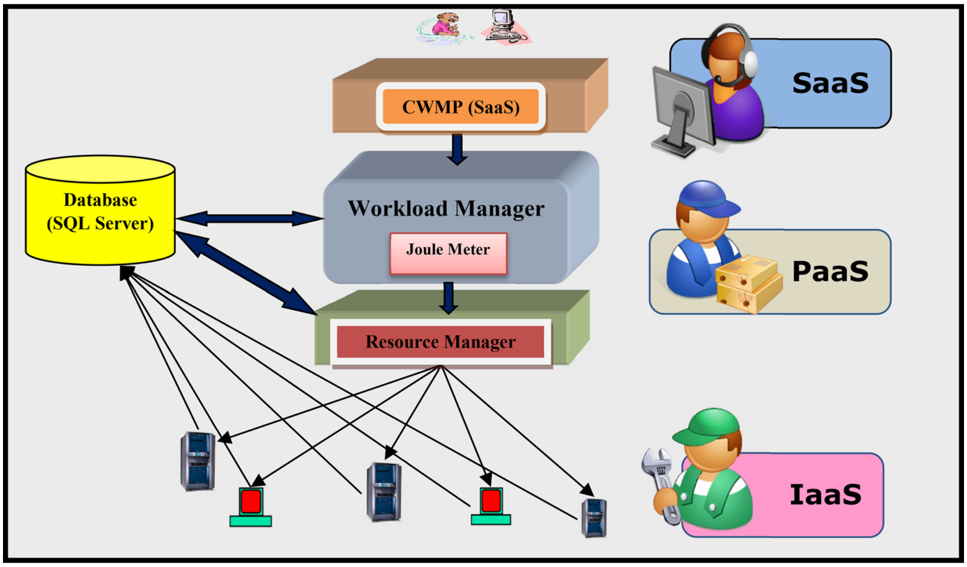


Table 6 Configuration details of testbed

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

be extended. Presently, toolkit is used for simulation of Cloud environment containing distinct and inter-networked Clouds. Furthermore, it provides custom interfaces to implement resource scheduling techniques for VM allocation under inter-networked circumstances in Cloud computing. The core benefits of this toolkit are used to test the performance along with time effectiveness (it needs very small time and effort to implement resource scheduling test environment for Cloud based application) and applicability and flexibility (with little development and programming effort, developer can test the performance of Cloud-based application in heterogeneous Cloud environments (Microsoft Azure, Amazon EC2)). For experimental results heterogeneous cloud workloads are considered. Each resource comprise different kind of machines, machine might have one or more than one PE (Processing Element) with different Million Instructions Per Second (MIPS). In this outcome, we suppose that each cloud workload which is admitted to the CHOPPER may need fluctuating input size and execution time of workload and such type of cloud workloads in the form of Cloudlets are described [29–31]. At *Infrastructure level*, three different servers (consist of virtual nodes) have been created through Citrix Xen Server and SQL Server has been used for data storage. Scheduler runs at IaaS level on Citrix Xen Server. Computing nodes used in this experiment work are further categorized into three categories as shown in Table 6.

The execution cost is calculated based on user workload and deadline (if deadline is too early (urgent) it will be more costly because we need a greater processing speed and free resources to process particular workload with urgency). There individual price is fixed (artificially) for different resources because all the resources are working in coordination to fulfill the demand of user (demand of user is changing dynamically). Experiment setup using three servers in which further virtual nodes [12 = 6 (Server 1) +4 (Server 2) +2 (Server 3)] are created. Every virtual node has different number for execution components (ECs) to process user request and every EC has their own cost [(C$/EC time unit (sec)]. Table 6 shows the characteristics of the resources used and their execution component (EC) access cost per time unit in Cloud dollars (C$) 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.

4.1 CHOPPER execution

The aim of performance evaluation is to demonstrate that it is feasible to implement and deploy the autonomic resource management approach on real cloud resources. Nodes in this system can be added or removed at runtime by autonomic resource manager based on the requirement. The key components of the cloud environment are: user interface (CWMP), CloudSim, SNORT, resource scheduler and resources. A detailed discussion of the implementation using CloudSim can be found in [31]. However, following details enable the understanding of the cloud based environment in which CHOPPER is implemented:

(1) Cloud consumer submit their request to user interface (CWMP) that contains the workload description [workload name, workload type, budget, deadline and resource scheduling policy (cost based or time based)].
(2) CWMP is deployed on CloudSim Toolkit (used as a scalable cloud middleware to make interaction between SaaS and IaaS).
(3) Resource configuration is identified to schedule the number of workloads based on QoS requirements as described by cloud consumer in the form of SLA.
(4) CHOPPER executes the different number of workloads using resources automatically as discussed in Sect. 3.8.
(5) Autonomic resource manager uses *Sensors* to measure the performance of system in terms of QoS to avoid violation of SLA and updated information is exchanged between all the autonomic units through *Effector*.
(6) CHOPPER reacts when a workload fails during its execution and self-healing capability of CHOPPER is used to re-negotiate the SLAs using WS-Agreement standard (Sect. 3.7) based on availability of resources (reserve resources can be used in case unavailability of resources in resource pool with different SLA) is shown in Fig. 2.
(7) After successful execution of workloads, this further returns the resources to resource pool as shown in Fig. 2.
(8) At the end, the autonomic unit returns updated workload's execution information along with processed workload back to the cloud consumer.

Table 7 Types of workload and their urgency details

Type of workload	Deadline urgency (D_u)	Slack time (seconds)	Delay time (seconds)	Deviation status	Minimum penalty	Penalty rate
Critical (urgent deadline)	$D_u < 0.25$	10	0–50	5%	200 Seconds	5%
			51–100	10%	400 s	6%
			101–150	15%	600 s	7%
Good (average deadline)	$0.25 \leq D_u \leq 0.75$	30	0–50	5%	100 s	4%
			51–100	10%	200 s	5%
			101–150	15%	300 s	6%
Normal (relaxed deadline)	$D_u > 0.75$	60	0–50	5%	50 s	2%
			51–100	10%	100 s	3%
			101–150	15%	150 s	4%

To validate CHOPPER, we have implemented a real application and presented performance evaluation through a web service i.e. *cloud workload management portal* by considering QoS parameters at service level as mentioned in [32].

4.2 Experimental results

All experiments were started with workload name: performance testing [processing larger image file of size 713 MB] as described in Table 5. Performance of CHOPPER has been evaluated in two steps: (i) case study of different type of workloads and (ii) validation of CHOPPER with existing autonomic resource management techniques.

4.2.1 Case study: type of workloads

In this research work, workloads are classified into three categories based on deadline urgency: (i) critical, (ii) good and (iii) normal. Equation 11 is used to calculate deadline urgency. Table 7 describes the details of type of workloads along with deadline details to provide compensation in case of missing deadline. In this research work, the value of threshold energy ($E_{threshold}$) is 176 kWh for *critical* workload, 144 kWh for *good* workload and 122 kWh for *normal* workload. Experiment has been conducted with different number of resources (6–36).

For example, calculating the compensation in which delay time = 50 (deviation status = 5%) seconds for "CRITICAL" workload is as following:

$$Compensation = Penalty_{minimum} + [Penalty\ Rate \times Delay\ Time]$$

$$Compensation = 200s + [5 \times 50\ s] = 450\ s$$

It will provide 450 s free cloud service as a compensation. Table 8 describes the details of type of workloads along with their QoS value.

In this experiment, four different cloud infrastructures with different processor configurations (2 core processor, 4 core processor, 8 core processor and 16 core processor) have been considered to measure the variation of QoS value. CHOPPER processes and converts a larger image file (Size = 713 MB) from JPEG format to PNG format. The conversion of a single JPEG file into PNG is considered as a single workload. 500 workloads of this type have been processed and executed to find the experiment statistics. Table 8 clearly shows that execution time of different type of workloads in 16 core processor is 28.53% lesser than 2 core processor, 22.11% lesser than 4 core processor and 18.36% lesser than 8 core processor. Execution time decreases with increase in number of cores of processor. Execution cost and energy consumption of different type of workloads increases with increase in processor configuration from 2 to 16 core processor because 16 cores needs larger cooling capacity as compared to 2 core processor. On the other side, throughput and resource utilization also increases from lower processor configuration to higher but energy efficiency reduces slightly. In 2 core processor, number of missed deadlines is more in critical workloads than normal and good while number of missed deadlines is lesser in critical workloads as compared to normal and good in 16 core processor because critical workloads getting priority for execution. Initially, SLA violation rate is 2.33% more in 2 core processor for critical workloads while it is 6.67% lesser in 16 core processor. With the increase in processor configuration, the resource contention is also increasing but average resource contention for critical workload is 16.11% lesser than normal and good workload. Intrusion detection rate is almost same for every configuration but fault detection rate is increasing from 2 core processor to 16 core processor. With the increase in processor configuration, availability and reliabil-

Table 8 Summary of experiment statistics of different type of workloads and their QoS value

QoS parameter	2 Core processor			4 Core processor			8 Core processor			16 Core processor		
	CRITICAL	GOOD	NORMAL	CRITICAL	GOOD	NORMAL	CRITICAL	GOOD	NORMAL	CRITICAL	GOOD	NORMAL
Execution time (sec)	125.44	140.66	153.44	108.64	110.88	118.72	101.92	109.76	115.36	95.2	99.68	104.16
Execution cost (C$)	97.44	95.2	90.72	113.12	108.64	103.04	127.68	124.32	120.96	143.36	135.52	132.16
Throughput (workload/sec)	500.64	449.12	454.72	549.92	516.32	520.81	617.12	573.44	582.4	684.32	719.04	675.36
Energy consumption (kWh)	71.68	62.72	42.56	80.64	76.16	57.12	98.56	90.72	84.21	115.36	110.88	106.4
Resource utilization (%)	82.88	80.64	81.76	87.36	84.69	79.52	92.96	87.36	84.23	98.56	99.68	97.44
Energy efficiency (%)	94.08	87.36	79.52	98.22	94.08	95.2	91.84	88.48	90.72	87.36	89.6	91.84
SLA violation rate (%)	7.48	6.72	5.6	12.32	11.2	13.44	16.8	15.68	19.04	23.52	25.76	24.41
No. of missed deadlines	8	7	5	7	7	6	6	8	7	5	8	8
Resource contention (Sec)	1792.12	1736.96	1927.52	3590.72	4986.24	5065.76	5401.76	5875.52	6547.52	6845.44	7756	7520.8
Intrusion detection rate (%)	20.16	19.04	20.16	24.64	25.76	28.63	16.8	15.68	13.44	19.04	20.16	23.52
Fault detection rate (%)	59.36	58.24	58.8	62.72	60.48	61.04	68.32	71.68	71.12	84.96	85.12	83.552
Availability (%)	91.84	94.08	92.96	91.84	91.84	94.08	95.2	96.32	99.68	100.8	101.92	103.712
Reliability (%)	12.32	13.0928	14.168	13.6752	14.168	13.8432	14.56	16.128	16.912	19.3984	19.2752	19.6448
Waiting time (sec)	248.64	259.84	274.4	244.16	255.36	253.12	212.8	221.76	231.84	180.32	200.48	207.2
Turnaround time (sec)	712.32	726.88	750.41	744.81	750.43	744.81	651.84	751.52	740.32	636.16	677.6	687.68

ity is also increasing but waiting time and turnaround time is decreasing. Table 8 reported that CHOPPER mainly focuses on critical workloads due to urgent deadline constraint to improve user satisfaction. Further, the variation of five important QoS parameters (energy consumption, average execution cost, execution time, resource utilization and SLA violation rate) is measured with different type of workloads (critical, good and normal).

Test Case 1: energy consumption: We have calculated value of energy consumption in kWh for different type of workloads (critical, good and normal) with different number of resource. With the increasing number of resources, the value of energy consumption also increases. The minimum value of energy consumption is 46.1 kWh at 6 resources as shown in Fig. 7. Critical workloads consume 7.73 and 12.75% more energy than good and normal workloads respectively. The maximum value of energy consumption is 206.53 kWh in critical workloads.

Test Case 2: average cost: We have used (Eq. 4–8) to calculate average cost. With the increase in number of resources, execution cost rises as shown in Fig. 8. The minimum value of cost is 72.6 C$ at 6 resources and maximum is 248.92 C$ at 36 resources. Critical workloads use 4.43 and 10.46% cost more than good and normal workloads respectively.

Test Case 3: execution time: We have used (Eq. 1) to calculate execution time. As shown in Fig. 9, the execution time decreases with increase in number of resources. At 6 resources, critical workload executes same number of workloads with 1.91 and 2.11% more execution time than good and normal workloads respectively. With 15–60 resources, execution time varies in same ratio but variation decreases abruptly after 24 resources.

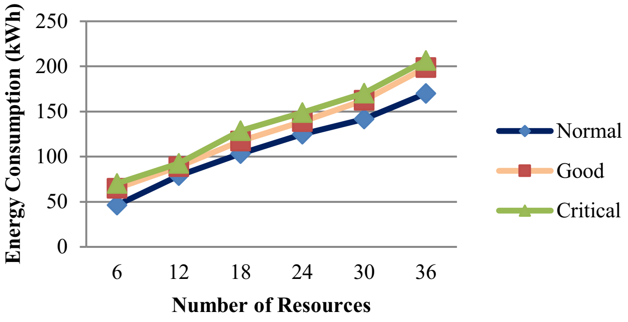


Fig. 7 Effect of change in number of resources submitted on energy consumption

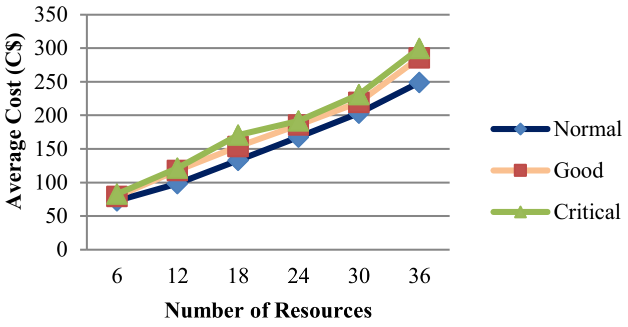


Fig. 8 Effect of average cost with change in number of resources

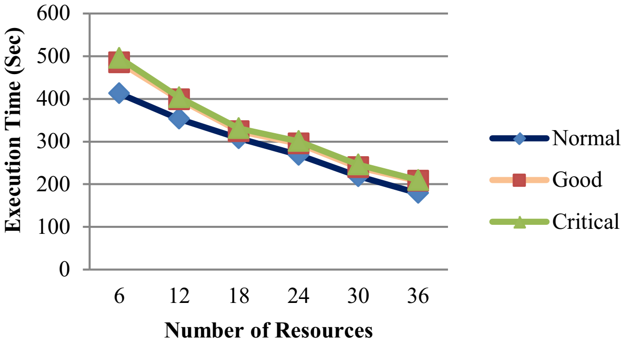


Fig. 9 Effect of execution time with change in number of resources

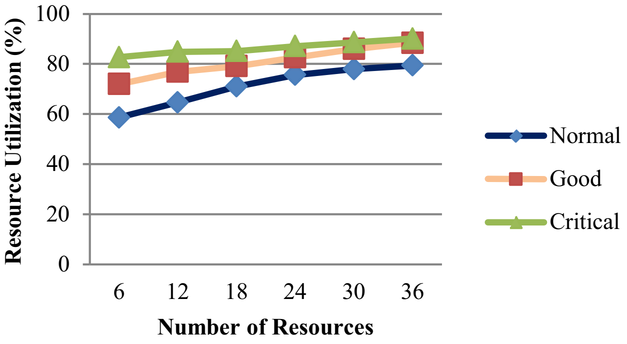


Fig. 10 Effect of change in number of resources submitted on resource utilization

Test Case 4: resource utilization: It is a ratio of actual time spent by resource to execute workload to total uptime of resource for that resource. We have used following formula to calculate resource utilization (Eq. 12).

$$Resource\,Utilization_i = \sum_{i=1}^{n} \left(\frac{actual\ time\ spent\ by\ resource\ to\ execute\ workload}{total\ uptime\ of\ resource} \right) \tag{12}$$

Where n is number of workloads. With increasing the number of resources, the percentage of resource utilization is increasing as shown in Fig. 10. Initially, resource utilization is more while executing critical workloads at 6 resources. The percentage of resource utilization in critical workload is 4.84 and 8.91% more than good and normal workloads respectively. The maximum value of resource utilization is 90.11% for execution of critical workloads. Figure 10 depicts the resource utilization as per workload requirement. Hence as the number of workloads increase so does the number

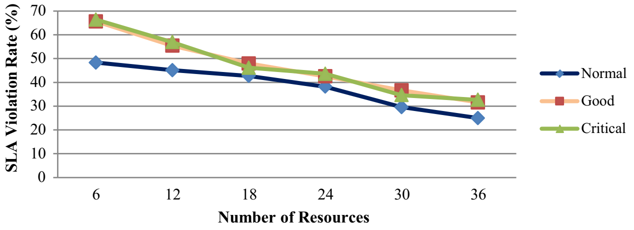


Fig. 11 Effect of change in number of resources on SLA violation rate

of resources. To a certain extent as the number of resources grow the resource utilization shall increase but in case if it reaches above the critical value the resource utilization will stale.

Test Case 5: SLA violation rate: It is defined as the product of Failure rate and weight of SLA [37]. We have used following formula to calculate SLA violation rate (Eq. 13). List of SLA = $< m_1, m_2 \ldots\ldots\ldots\ldots\ldots.m_n >$, where n is total number of SLAs.

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

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

$$SLA\ Violation\ Rate\ = \ Failure\ Rate \times \sum_{i=1}^{n} (w_i) \qquad (13)$$

Where $w_i$is weight for every SLA. We have analyzed the effect of change in number of resources on SLA violation rate. SLA violation rate is changed with different number of resources as shown in Fig. 11. Value of SLA violation rate is varied between 0 and 100%. At 6 resources, SLA violation rate is 2.27 and 11.44% more than good and normal workloads. SLA violation rate is suddenly decreased at 24 resources. SLA violation rate at 36 resources for normal workload is 7.73 and 8.23% than good and critical workloads respectively.

4.2.2 Validation of CHOPPER with Existing Techniques

We have verified CHOPPER for all four aspects: self-optimizing, self-healing, self-protecting and self-configuring. We have used different metrics for verification of performance of CHOPPER [4–6,32,36,37]. We have performed the different number of experiments in different type of verification by comparing CHOPPER with existing autonomic resource management techniques. Experiment has been conducted with different number of workloads (500–3000) for verification of different QoS parameters and performance of all the QoS parameters has been evaluated. The following existing resource management approaches have been considered to validate CHOPPER:

(i) ***PURS:*** Jose and Luıs [35] proposed a partial utility-driven resource scheduling (PURS) technique for elastic SLA and pricing negotiation which permits providers exchanging resources between VMs in expressive and economically effective ways. Further, a comprehensive cost method is defined by including partial utility given by customers to a definite level of degradation, when VMs are assigned in overcommitted situations. In this technique, revenue per resource allocation and execution time is improved.

(ii) ***ARCS***: Mehdi et al. [23] 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 has 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.

(iii) ***SHAPE***: Self-healing and self-protection environment (SHAPE) [26] 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.

(iv) ***EARTH***: Energy-aware autonomic resource scheduling (EARTH) [6] 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.

(v) ***QRPS***: QoS based resource provisioning and scheduling (QRPS) framework [33] is used for resource provisioning in which: (i) clustering of workloads is done through workload patterns, (ii) k-means based clustering algorithm is used for re-clustering of workloads after assigning weights to quality attributes of each workload and (iii) QoS requirements of clustered workloads are identified and resources are provisioned by resource provisioner based on their QoS requirements. Further, four different resource scheduling policies (cost, time, cast-time and bargaining based) are used to schedule the provisioned resources based on QoS requirements described by cloud consumer through SLA.

We used *SHAPE* [26] to evaluate the performance of CHOPPER in terms of fault detection rate, intrusion detection rate, reliability, availability and waiting time, used *EARTH* [6] to evaluate the performance of CHOPPER in terms of energy efficiency, energy consumption and resource utilization, used *PURS* [35] to evaluate the performance of CHOPPER in terms of SLA violation rate, used *ARCS* [23] to evaluate the performance of CHOPPER in terms of resource contention, and used *QRPS* [33] to evaluate the performance of CHOPPER in terms of execution cost, execution time and throughput. 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) in this research work are described in our previous work [4–6,33,36,37].

Self-optimizing verification: experiment has been conducted with different number of workloads (500–3000) for verification of self-optimizing aspect. We have calculated execution cost, energy efficiency, execution time and resource contention. For verification of their characteristics of CHOPPER self-optimization includes:

Test Case 1—execution cost It is defined as the total amount of cost spent per one hour for the execution of workload and measured in cloud dollars (C$). We have used following formula to calculate execution cost (Eq. 14).

$$Execution\ Cost = \frac{Total\ Amount\ of\ Cost\ Spent}{Hour} \tag{14}$$

With the increase in number of workloads, execution cost rises as shown in Fig. 12. As the number of workload increases, CHOPPER performs better QRPS. The cause is that CHOPPER adjusts the resources at runtime according to the QoS requirements of workload. With the increase in number of workloads, resource utilization increases as shown in Fig. 13. Utilization of resources increases due to increase in number of workloads, to execute more number of resources as required. At 66% level of resource utilization, execution cost is 34–39% lesser in CHOPPER than QRPS but at 96% level of resource utilization, execution cost is 6.2–8% lesser in CHOPPER than QRPS. Execution cost suddenly increases at the 84% resource utilization level but CHOPPER performs better than QRPS.

Test Case 2—energy efficiency It is a ratio of number of workloads successfully executed in a data center to total energy consumed (E_{Cloud}) to execute those workloads [36].

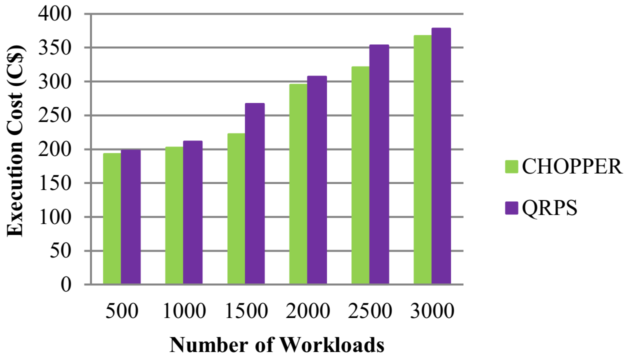


Fig. 12 Effect of execution cost with change in number of workloads

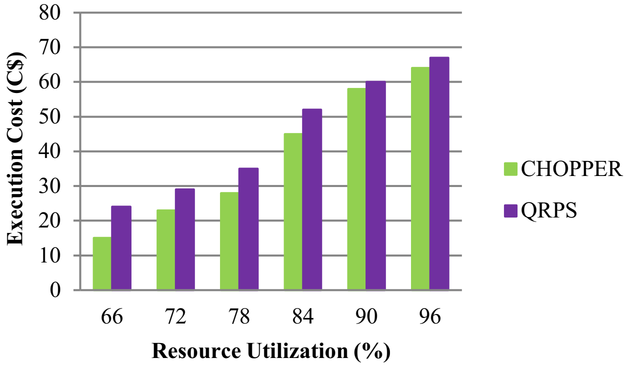


Fig. 13 Effect of execution cost on resource utilization

We have used following formula to calculate energy efficiency (Eq. 15).

$$Energy\ Effieviency = \sum_{i=1}^{n} \left(\frac{number\ of\ workloads\ successfully\ executec\ in\ a\ data\ center}{total\ energy\ consumed\ to\ executed\ those\ workloads} \right) \tag{15}$$

Where n is the number of workloads to be executed. With increasing number of cloud workloads, the value of energy efficiency is decreases. The value of energy efficiency in CHOPPER is more as compared to EARTH at different number of cloud workloads as shown in Fig. 14. The maximum value of energy efficiency is 89.8% at 1000 cloud workloads in CHOPPER.

Test Case 3—execution time We have used (Eq. 1) to calculate execution time. As shown in Fig. 15, the execution time increases with increase in number of workloads. At 2000 workloads, execution time in CHOPPER is 12.94% lesser than QRPS. After 2000 workloads, execution time increases abruptly but CHOPPER performs better than QRPS. We have calculated value of execution time for both CHOPPER and QRPS at different level of resource utilization as shown in Fig. 16. At 66% level of resource utilization, execution time

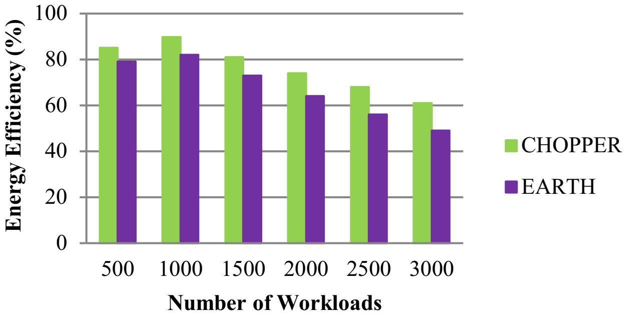


Fig. 14 Effect of energy efficiency with change in number of workloads

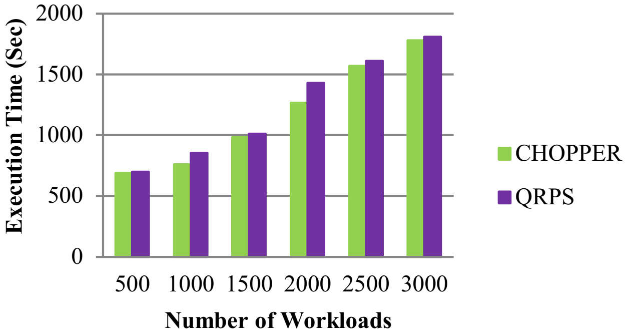


Fig. 15 Effect of execution time with change in number of workloads

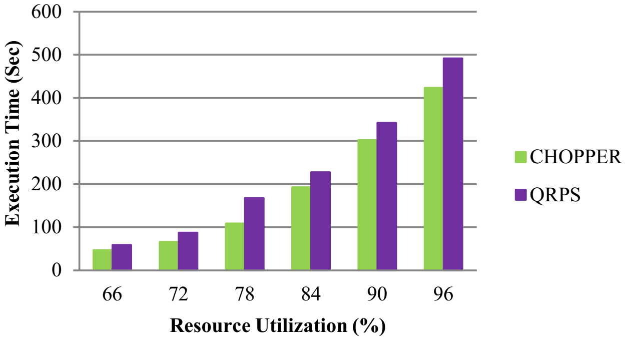


Fig. 16 Effect of execution time on resource utilization

in CHOPPER is 5.33% lesser than QRPS but at 96% level of resource utilization execution time is 16.7% lesser.

Test Case 4—resource contention When more than one workload shares same resource then resource contention may occur [23]. It occurs due to following reasons: (i) when more than one workload executing on same resource, (ii) more number of workloads can create more resource contention and (iii) if the number of provided resources are lesser than the number of required resources. Resource contention ($ResCon$) is defined during scheduling of resources at time t. We have used following formula to find resource contention (Eq. 16).

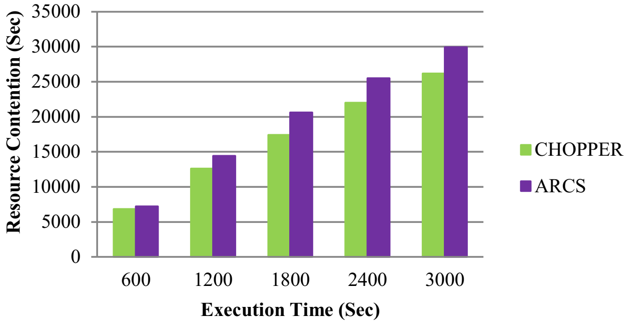


Fig. 17 Effect of resource contention on time

$$ResCon\,(t) = \sum_{r \in ResourceList} ResCon\,(t, r)$$

$$ResCon\,(t, r) = \sum_{rt \in ResourceType} ResCon\,(t, r, rt)$$

$$RCStatus(t, r, rt) = \begin{cases} 1, & \sum\limits_{w \in wS(t,r)} (rt \in w.OVERLOAD == TRUE?1:0) > 1 \\ 0 & otherwise \end{cases}$$

$$RCStatus(t, r, rt) = \begin{cases} \sum\limits_{w \in wS(t,r) \wedge rt \in w.OVERLOAD} w.\mathrm{R}esourceRequirment[rt], RCStatus(t,r,rt)=1 \\ \qquad otherwise=0 \end{cases} \tag{16}$$

where ***r*** is list of resources, ***rt*** is used to specify the type of resource, $w.OVERLOAD$ is used to specify the workload ***w*** overloads the resource, $w.ResourceRequirment$ is used to specify the resource requirement of ***w*** in terms of capacity (memory, processor etc.) and $RCStatus\,(t, r, rt)$ specify the current status of resource contention in terms of Boolean statements [true or false]. Throughout all experiments this value, measured in seconds, is as a value for comparison, and not an exact time for resource contention. We have calculated the value of resource contention for both CHOPPER and ARCS at different level of execution time as shown in Fig. 17. Value of resource contention increases with increase in execution time. Resource contention at 600 s is minimum (in CHOPPER is 5.95% lesser than ARCS and maximum at 3000 seconds (in CHOPPER is 14.28% lesser than ARCS). We have also analyzed the effect of resource contention on number of workloads as shown in Fig. 18. With increase in number of workloads, value of resource contention increases from 500 workloads to 3000 workloads. Resource contention at 500 workloads in CHOPPER is 11.91% lesser than ARCS and at 3000 workloads in CHOPPER is 13.51% lesser than ARCS. From 1000 workloads to 2500 workloads, value of resource contention is almost stable in both CHOPPER and ARCS, but CHOPPER performs better.

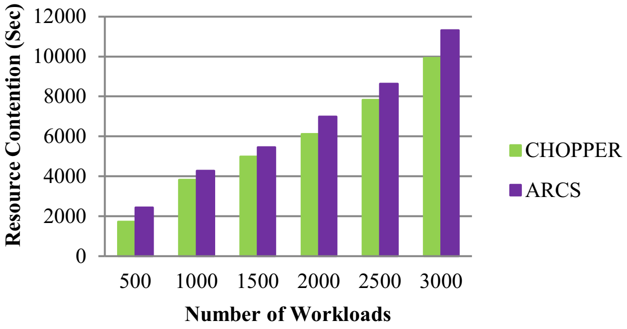


Fig. 18 Effect of resource contention with change in number of workloads

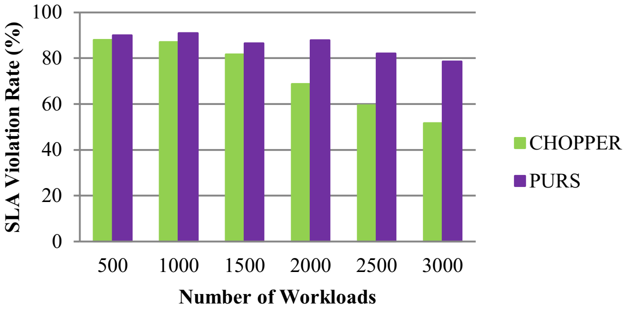


Fig. 19 Effect of change in number of workloads on SLA violation rate

Test Case 5—SLA violation rate: We have used (Eq. 13) to calculate SLA violation rate. We have analyzed the effect of change in number of workloads on SLA violation rate. SLA violation rate is changed with different number of workloads as shown in Fig. 19.

Variation in SLA violation rate in PURS is larger as compared to CHOPPER. Value of SLA violation rate is varied between 0 and 100%. At 500 workloads, SLA violation rate in CHOPPER is 2.27% lesser than PURS but SLA violation rate suddenly decreases at 2000 workloads. SLA violation rate in CHOPPER at 2000 workloads is 11.71% lesser than PURS but at 3000 workloads, SLA violation rate is 16.38% lesser. There are different numbers of deadlines missed in different techniques as shown in Fig. 20. With increasing the number of cloud workloads, the number of missed deadlines also increases. The number of missed deadlines in PURS is more than CHOPPER as shown in Fig. 20. The variation in number of deadlines missed at 500 and 1500 workloads is lesser as compared to the 2000 and 2500 workloads but there is maximum variation at 3000 workloads i.e. as workloads no increases.

Self-healing verification Experiment has been conducted with different number of workloads (500–3000) for verification of self-healing. We have calculated fault detection rate, throughput, reliability, availability, waiting time and turnaround time. For verification of their characteristics of CHOPPER self-healing includes:

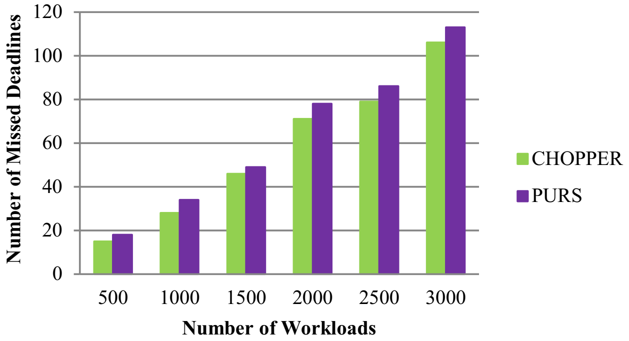


Fig. 20 Effect of change in number of workloads on number of missed deadlines

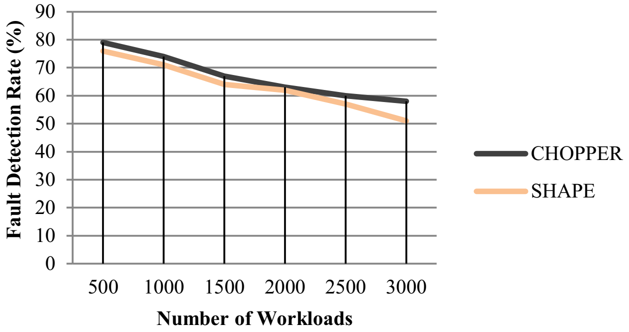


Fig. 21 Fault detection rate vs. number of workloads

Test Case 1—fault detection rate: It is the ratio of number of faults detected to the total number of faults existing. Faults may be software or hardware. We have used following formula to calculate fault detection rate (Eq. 17).

$$Fault\ Detection\ Rate = \frac{Number\ of\ Faults\ Detected}{Total\ number\ of\ Faults} \tag{17}$$

Figure 21 shows the capability of CHOPPER to detect the failures by injecting different number of faults in the system with different number of workloads. Fault detection rate decreases with increase in number of workloads. From 500 workloads to 1500 workloads, value of fault detection rate reduces in both CHOPPER and SHAPE, but CHOPPER performs better than SHAPE. At 2000 workloads, fault detection rate is almost same for both the techniques but at 3000 workloads fault detection rate in CHOPPER is 13.72% more than SHAPE.

Test Case 2—throughput: It is the ratio of total number of workloads to the total amount of time required to execute the workloads. We have used following formula to calculate throughput (Eq. 18).

$$Throughput = \frac{\text{Total Number of Workloads } (W_n)}{\text{Total amount of time required to executethe workloads } (W_n)} \tag{18}$$

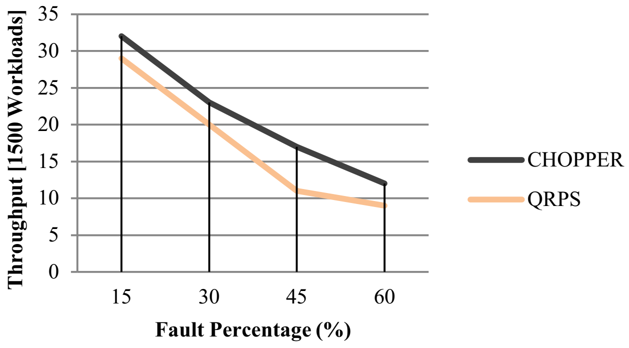


Fig. 22 Throughput [1500 workloads] vs. fault percentage (%)

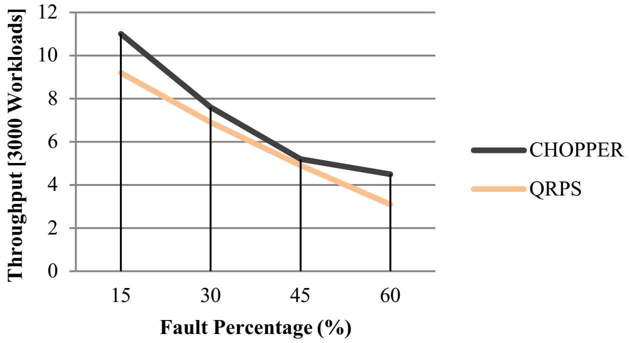


Fig. 23 Throughput [3000 workloads] vs. fault percentage (%)

We have injected number of software, network and hardware faults (fault percentage) to verify the throughput of the CHOPPER with different number of workloads (1500 and 3000). Figure 22 shows the comparison of throughput of both CHOPPER and QRPS at 1500 workloads and it is clearly shown that CHOPPER performs better than QRPS. In our experiment, we found the maximum value of throughput at fault percentage 45% i.e. CHOPPER has 26% more throughput than QRPS. Figure 23 shows the comparison of throughput of both CHOPPER and QRPS at 3000 workloads and it is clearly shown that CHOPPER performs better than QRPS. In our experiment, we found the maximum value of throughput at fault percentage 15 and 60% but minimum at 45% i.e. CHOPPER has only 3.26% more throughput than QRPS.

Test Case 3—availability: It is a ratio of Mean Time Between Failure (MTBF) to Reliability. We have used following formula to calculate availability (Eq. 19).

$$Availability = \frac{MTBF}{MTBF + MTTR} \tag{19}$$

Test Case 4—reliability: It is an addition of Mean Time Between Failure (MTBF) and mean time to repair (MTTR). We have used following formula to calculate reliability (Eq. 20).

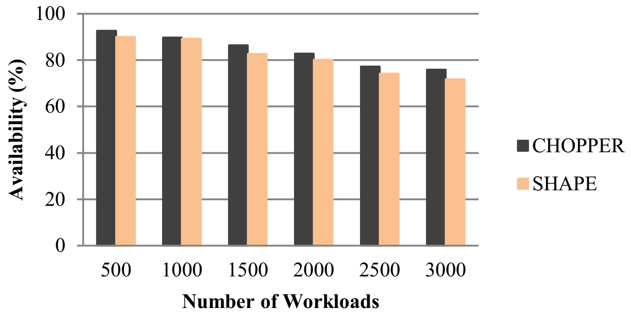


Fig. 24 Availability vs. number of workloads

$$Reliability = MTBF + MTTR \tag{20}$$

Where mean time between failure (MTBF) is ratio of total uptime to number of breakdowns (Eq. 21).

$$MTBF = \frac{Total\ Uptime}{Number\ of\ Breakdowns} \tag{21}$$

Where mean time to repair (MTTR) is ratio of total downtime to number of breakdowns [Eq. 22].

$$MTTR = \frac{Total\ Downtime}{Number\ of\ Breakdowns} \tag{22}$$

We have calculated percentage of availability for both CHOPPER and SHAPE. With increasing the number of cloud workloads, the percentage of availability decreases. The percentage of availability in CHOPPER is more as compared to SHAPE at different number of cloud workloads as shown in Fig. 24. The maximum percentage of availability is 92.6% at minimum number of cloud workloads. By increasing the number of resources, the percentage of availability increases. CHOPPER performs better than SHAPE in terms of availability at different number of resources as shown in Fig. 25. The maximum percentage of availability is 91.7% at maximum number of resources. We have calculated percentage of reliability for both CHOPPER and SHAPE.

By increasing the number of cloud workloads, the percentage of reliability decreases but difference of reliability of two techniques is larger at 3000 workloads. The percentage of reliability in CHOPPER is more as compared to SHAPE at different number of cloud workloads as shown in Fig. 26. The maximum percentage of reliability is 9% at 500 workloads. By increasing the number of resources, the percentage of reliability increases. CHOPPER performs better than SHAPE in terms of reliability at different number of resources as shown in Fig. 27. The maximum percentage of reliability is 12.3% at 250 resources in CHOPPER.

Test Case 5—waiting time: We have used (Eq. 2) to calculate waiting time. We have injected failures to verify the performance in terms of waiting time of workloads in

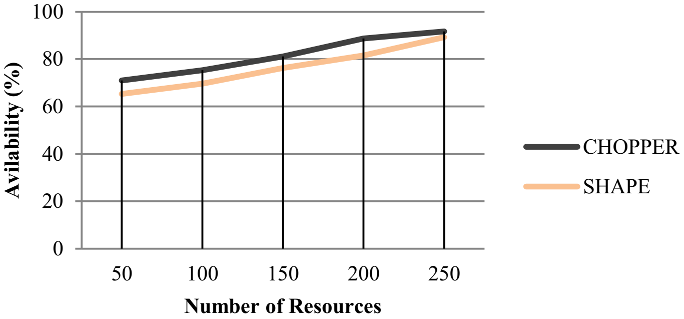


Fig. 25 Availability vs. number of resources

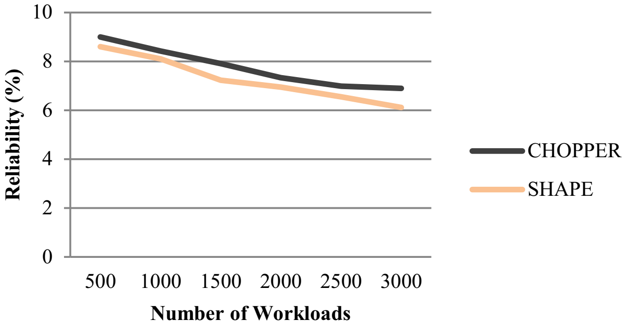


Fig. 26 Reliability vs. number of workloads

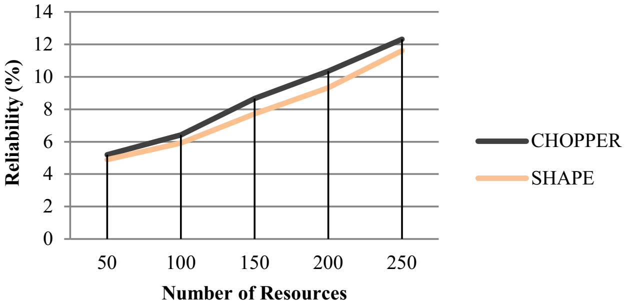


Fig. 27 Reliability vs. number of resources

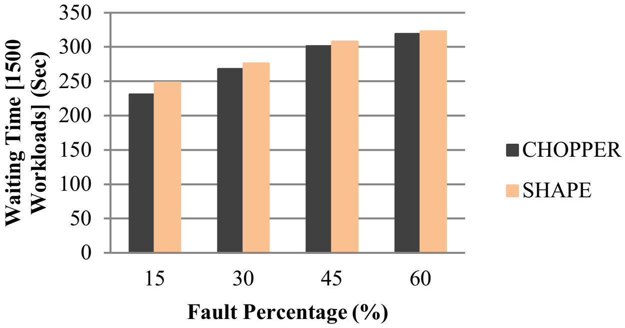


Fig. 28 Waiting time [1500 workloads] vs. fault percentage

CHOPPER with different fault percentage (15–60%). Figure 28 shows the comparison of waiting time of both CHOPPER and SHAPE at 1500 workloads and it is clearly shown that CHOPPER performs better than SHAPE. In our experiment, we found the maximum difference in waiting time with minimum fault percentage (15%) i.e. 7.35% and at 60% fault percentage, difference is just 1.25%. The comparison of waiting time of both CHOPPER and SHAPE at 3000 workloads is shown in Fig. 29 and it is clearly shown that CHOPPER performs better than SHAPE. Waiting time increases with increase in percentage of fault rate. In our experiment, we found the maximum difference in waiting time with fault percentage (45%) i.e. 3.82% and with other fault percentage, the waiting time is almost same but CHOPPER performs better.

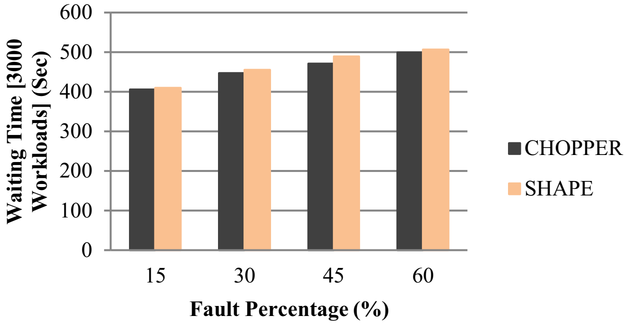


Fig. 29 Waiting time [3000 workloads] vs. fault percentage

Test Case 6—turnaround time: It is the ratio of difference of workload completion time (WC_i) and workload submission time (WS_i) to number of workloads. We have used following formula to calculate turnaround time (Eq. 23).

$$Turnaround\ Time_i = \sum_{i=1}^{n} \left(\frac{WC_i - WS_i}{\text{n}} \right) \tag{23}$$

where n is the number of workloads. To verify the performance in terms of turnaround time of workloads in CHOPPER with different fault percentage (15–60%), CHOPPER is deployed on different nodes. Figure 30 shows the comparison of turnaround time of CHOPPER and SHAPE at 1500 workloads and it is clearly shown that CHOPPER performs better than SHAPE. In our experiment, we found the maximum difference in turnaround time with fault percentage

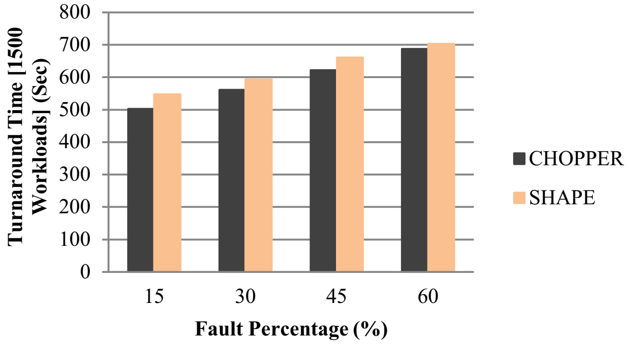


Fig. 30 Turnaround time [1500 workloads] vs. fault percentage

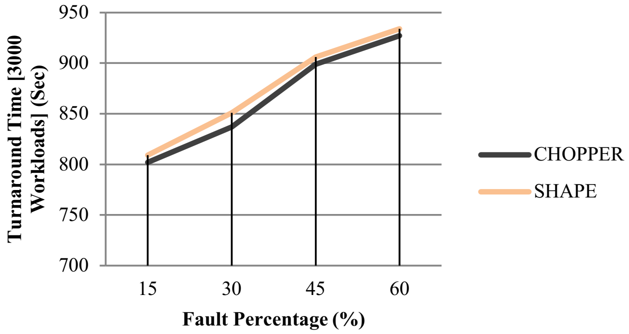


Fig. 31 Turnaround time [3000 workloads] vs. fault percentage

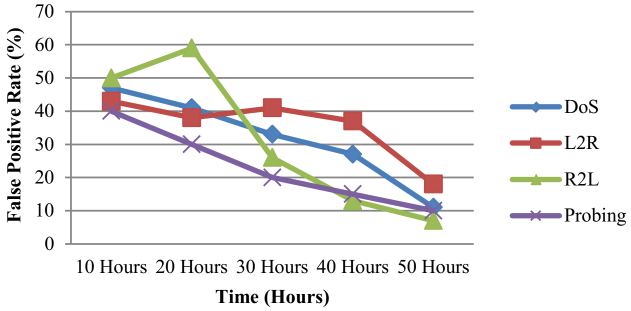


Fig. 32 False positive rate vs. time

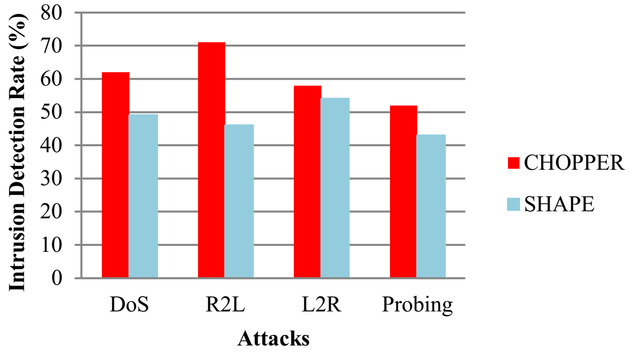


Fig. 33 Detection rate vs. attacks

(45%) i.e. 6.27 and at 60% fault percentage, difference is just 2.32%. The comparison of turnaround time of both CHOPPER and SHAPE at 3000 workloads is shown in Fig. 31 and it is clearly shown that CHOPPER performs better than SHAPE. Turnaround time increases with increase in percentage of fault rate. In our experiment, we found the maximum difference in turnaround time with fault percentage (30%) i.e. 1.67 % and with other fault percentage, the turnaround time is almost same but CHOPPER performs better than SHAPE.

Self-protecting verification Experiment has been conducted with different type of attacks (DoS, R2L, U2R and Probing) for verification of self-protecting. We have used different tools (metasploit framework for DoS, Hydra for R2L, NetCat for L2R and NMAP for probing) to launch different attacks [26]. We have calculated false positive rate and detection rate. For verification of characteristics of CHOPPER self-protecting includes:

Test Case 1—false positive rate: It is the ratio of false positives to the addition of false positives and true negatives. We have used following formula to calculate false positive rate (Eq. 24).

$$False\ Positive\ Rate = \frac{False\ Positives}{False\ Positives + True\ Negatives} \tag{24}$$

False positive rate decreases in CHOPPER with time and it is minimum at 50 hours as shown in Fig. 32. We have considered four types of attacks (DoS, R2L, U2R and Probing) and measured False Positive Rate for every attack. For R2L attack, False Positive Rate is higher as compared to other attacks.

Test Case 2—intrusion detection rate: It is the ratio of total number of true positives to the total number of intrusions. Detection rate increases with respect to time and it considers the number of blocked and detected attacks. For new attack or intrusion detection, database is updated with new signatures and new polices and rules are generated to avoid same attack. We have conducted experiment for known attacks; it is clearly shown in Fig. 33 that CHOPPER performs better than SHAPE (SNORT anomaly detector). We have removed signatures of some known attacks from database to verify the working of CHOPPER. We have used following formula to calculate detection rate (Eq. 25).

$$Intrusion\ Detection\ Rate = \frac{Total\ Number\ of\ True\ Positives}{Total\ Number\ of\ Intrusions} \tag{25}$$

Detection rate increases with respect to time as shown in Fig. 34. We have conducted an experiment for 144 h and we found that detection rate of CHOPPER is better than the SHAPE and performed better than earlier after 120 h. The variation of intrusion detection rate with different number of

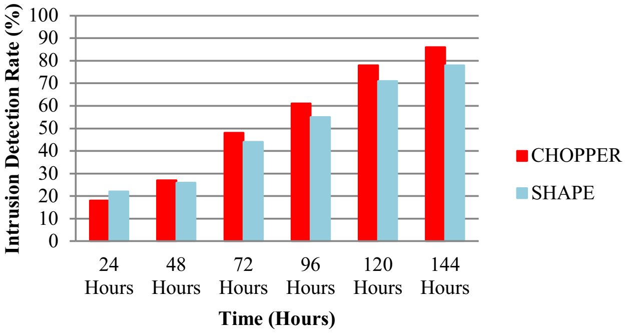


Fig. 34 Intrusion detection rate vs. time

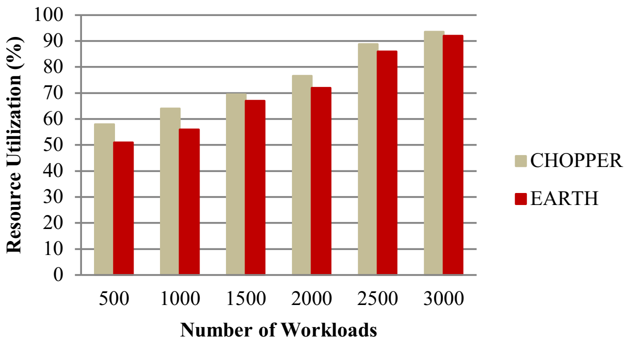


Fig. 36 Effect of change in number of workloads submitted on resource utilization

workloads and different type of attacks is shown in Fig. 35. With increasing the number of workloads, intrusion detection rate increases. It is clearly shown that the CHOPPER performs better in probing.

Self-configuring verification: experiment has been conducted with different number of workloads (500–3000) for verification of self-configuring. We have calculated resource utilization, average cost, SLA violation rate and energy consumption. For verification of characteristics of CHOPPER self-configuring includes:

Test Case 1—resource utilization: We have used (Eq. 11) to calculate resource utilization. With increasing the number of cloud workloads, the percentage of resource utilization increases. The percentage of resource utilization in CHOPPER is more as compared to EARTH at different number of cloud workloads as shown in Fig. 36. The maximum percentage of resource utilization is 93.61% at 3000 cloud workloads in CHOPPER but CHOPPER performs better than EARTH.

Test Case 2—average cost: We have calculated value of average cost using (Eqs. 4–8) for both CHOPPER and SHAPE with different number of cloud workloads as shown in Fig. 37.

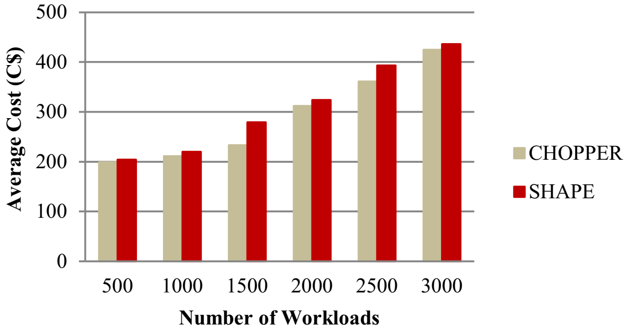
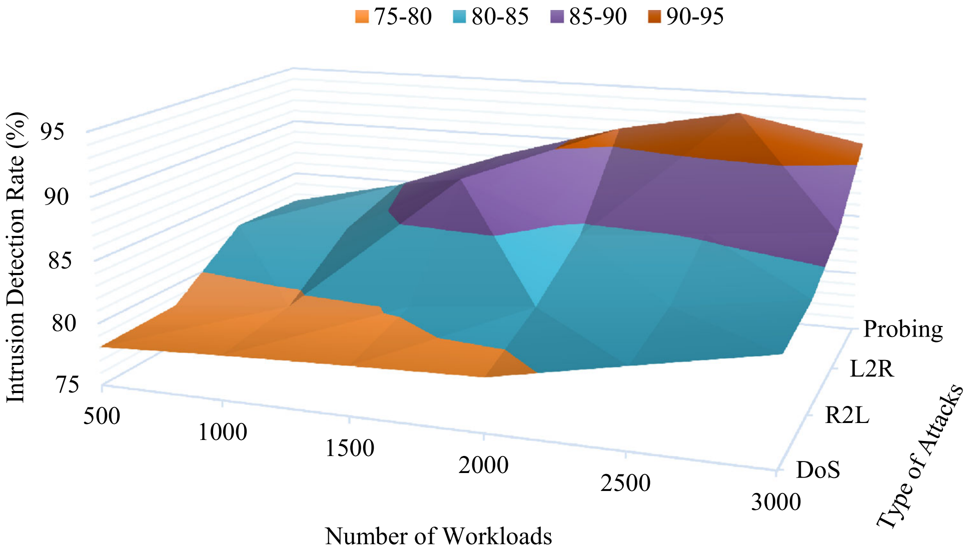


Fig. 37 Effect of change in number of workloads submitted on average cost

Average cost is an addition of resource cost and penalty cost. CHOPPER defined the different levels of penalty rate based on QoS requirements.

Delay time is difference of deadline and time when workload is actually completed. Average cost increases with increase in number of workloads. At 500 workloads, aver-

Fig. 35 Statistical analysis of intrusion detection rate



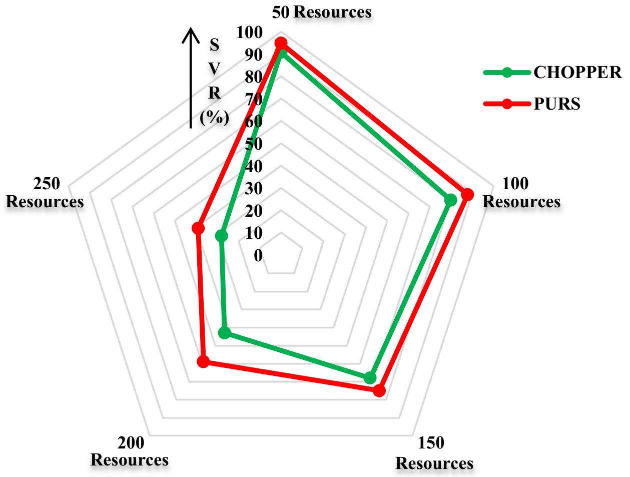


Fig. 38 Effect of change in number of resources on SLA violation rate (SVR)

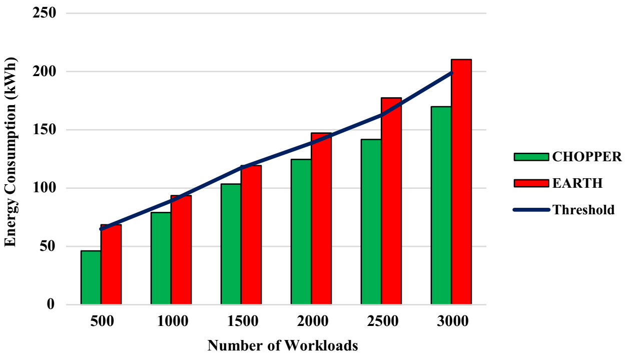


Fig. 39 Energy consumption vs. number of workloads

age cost in CHOPPER is slightly lesser than SHAPE but CHOPPER performs excellent at 1500 workloads. At 1500 workloads, average cost in CHOPPER is 19.74% lesser than SHAPE.

Test Case 3—SLA violation rate: We have used (Eq. 13) to calculate SLA violation rate (SVR). We have analyzed the effect of change in number of resources on SLA violation rate. SLA violation rate decreases with increase in number of resources as shown in Fig. 38. Variation in SLA violation rate in PURS is larger as compared to CHOPPER. Value of SLA violation rate is varied between 0 and 100%. At 50 resources, SLA violation rate in CHOPPER is 4.39% lesser than PURS but SLA violation rate is decreasing. At 200 resources, SLA violation rate in CHOPPER is 9.84% lesser than PURS but at 3000 workloads, SLA violation rate is 7.66% lesser. Experimental results show that the CHOPPER performs better than PURS.

Test Case 4—energy consumption: We have calculated value of energy consumption in kilo Watt hour (kWh) for CHOPPER and EARTH with different number of cloud workloads. With the increasing number of cloud workloads,

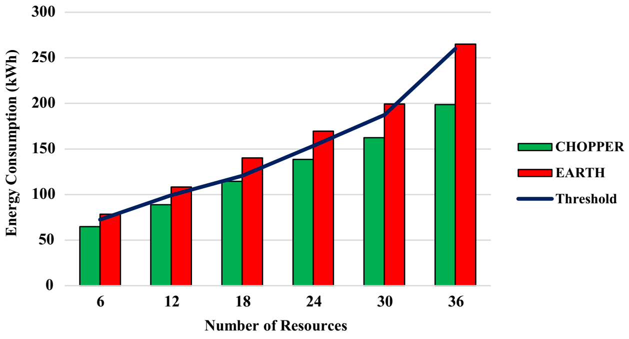


Fig. 40 Energy consumption vs. number of resources

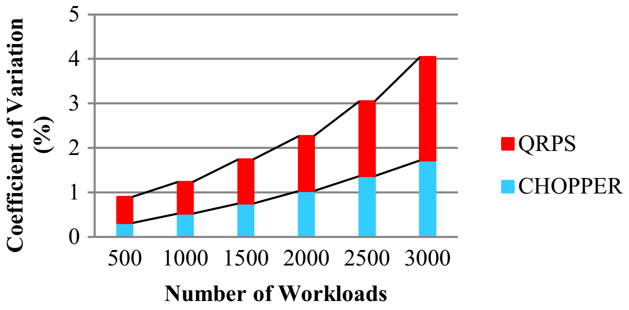


Fig. 41 Coefficient of variation for average execution cost with each resource management technique

the value of energy consumption also increases. The minimum value of energy consumption is 46.1 kWh at 500 workloads. CHOPPER performs better than EARTH in terms of energy consumption at different number of cloud workloads as shown in Fig. 39. It is clearly shown that the energy consumption in CHOPPER is always lesser than threshold value of energy consumption. The average value of energy consumption in CHOPPER is 16.66% lesser than EARTH.

Figure 40 shows the energy consumption for CHOPPER and EARTH with different number of resources. With the increasing number of resources, the value of energy consumption also increases. It is clearly shown that the energy consumption in CHOPPER is always lesser than threshold value of energy consumption. The average value of energy consumption in CHOPPER is 12.98% lesser than EARTH but at 30 resources, CHOPPER consumes 18.54% lesser energy as compared to EARTH.

4.3 Statistical analysis

Statistical significance of the results has been analyzed by Coefficient of Variation (*Coff. of Var.*), a statistical method. *Coff. of Var.* is statistical measure of the distribution of data about the mean value. *Coff. of Var.* is used to compare to different means and furthermore offer an overall analysis of performance of the CHOPPER used for creating the statistics. It states the deviation of the data as a proportion of its average

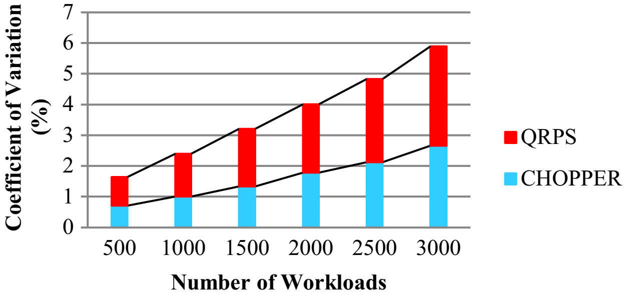


Fig. 42 Coefficient of variation for execution time with each resource management technique

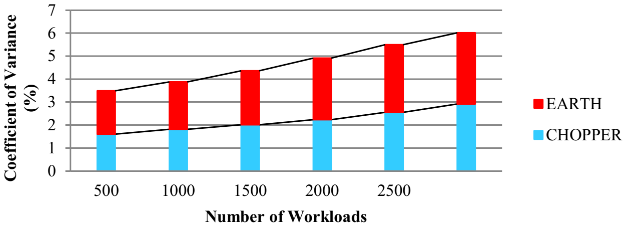


Fig. 43 Coefficient of variation for resource utilization with each resource management technique

value, and is calculated as follows (Eq. 26):

$$Coff.of\ Var. = \frac{SD}{\mathrm{M}} \times 100 \tag{26}$$

Where SD is a Standard Deviation and$M$ is Mean.

We used an average of thirty to fifty runs in order to guarantee statistical correctness. *Coff. of Var.* of average execution cost, execution time, resource utilization, energy consumption and SLA violation rate has been studied with respect to number of workloads for both CHOPPER and existing resource management technique (QRPS, EARTH and PURS) with different number of cloud workloads as shown in Figs. 41, 42, 43, 44 and 45 respectively.

Range of *Coff. of Var.* is (0.31–1.72%) for average execution cost and (0.71–2.66%) for execution time approves the stability of CHOPPER as shown in Figs. 41 and 42 respectively. Range of *Coff. of Var.* is (1.6–2.92%) for resource utilization and (1.12–3.69%) for energy consumption approves the stability of CHOPPER as shown in Figs. 43 and 44 respectively

Range of *Coff. of Var.* is (0.48–1.10%) for SLA violation rate with respect to number of workloads approves the stability of CHOPPER as shown in Fig. 45. Small value of *Coff. of Var.* signifies CHOPPER is more efficient in resource management in the situations where the number of cloud workloads has changed. Value of *Coff. of Var.* increases as the number of workloads is increasing. CHOPPER attained the best results in the cloud for average execution cost, execution time, resource utilization, energy consumption and SLA violation rate has been studied with respect to number of workloads.

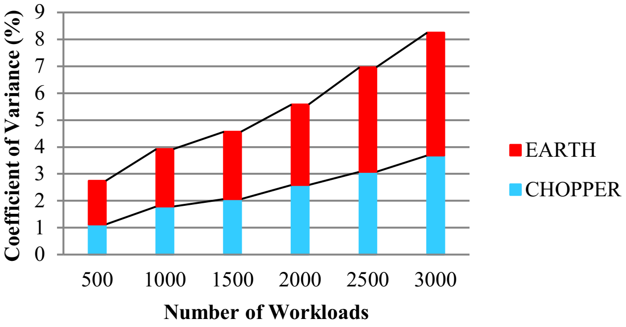


Fig. 44 Coefficient of variation for energy consumption with each resource management technique

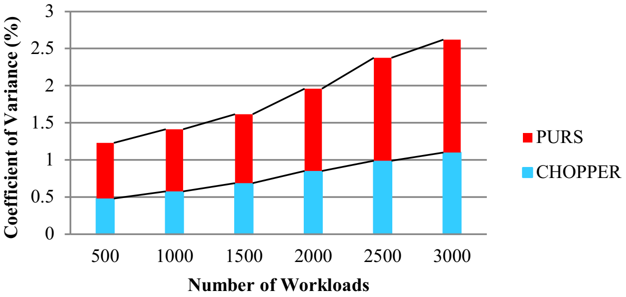


Fig. 45 Coefficient of variation for SLA violation rate with each resource management technique

4.4 Discussions

In this experiment, four different cloud infrastructures with different processor configurations (2 core processor, 4 core processor, 8 core processor and 16 core processor) have been considered to measure the variation of different QoS parameters such as fault detection rate, throughput, reliability, availability, waiting time, turnaround time, intrusion detection rate, resource utilization, SLA violation rate, execution cost, execution time, energy efficiency and resource contention. Table 9 describes summary of experiment statistics and percentage of Overall Improvement (OI) of different QoS parameters. 3000 workloads of same type (performance testing) have been considered on real cloud environment.

Existing autonomic resource management techniques such as SHAPE [26], EARTH [6], ARCS [23], PURS [35] and QRPS [33] are used to validate CHOPPER in five different cases. Small value of coefficient of variation signifies CHOPPER is more efficient and stable in resource management of the situations where the number of cloud workloads has changed. In all the different cloud infrastructures, experimental results demonstrate that CHOPPER improves the average intrusion detection rate by 11.52%, average fault detection rate by 3.62%, average energy efficiency by 10.92%, average resource utilization by 13.28%, average throughput by 5.82%, average availability by 2.84% and average reliability by 5.87% and it reduces the aver-

Table 9 Summary of experiment statistics and overall improvement (OI)

QoS parameter	2 Core processor			4 Core processor			8 Core processor			16 Core processor		
CASE-1	CHOPPER	QRPS	OI (%)	CHOPPER	QRPS	OI (%)	CHOPPER	QRPS	OI (%)	CHOPPER	QRPS	OI (%)
Execution time (sec)	178.68	207.91	14.06	218.65	259.71	15.81	307.66	378.56	18.73	412.65	577.67	28.57
Execution cost (C$)	72.65	79.31	8.46	98.23	125.64	21.82	167.72	219.54	23.60	247.69	285.66	13.29
Throughput (workload/sec)	547.54	502.25	8.96	593.89	561.16	5.70	661.45	622.56	6.27	703.56	687.56	2.33
CASE-2	CHOPPER	EARTH	OI (%)	CHOPPER	EARTH	OI (%)	CHOPPER	EARTH	OI (%)	CHOPPER	EARTH	OI (%)
Energy consumption (kWh)	46.11	64.98	28.97	88.91	117.61	24.40	124.46	162.13	23.22	170.68	212.36	19.63
Resource utilization (%)	71.96	58.62	22.76	79.11	71.01	11.41	83.66	77.761	7.59	88.41	79.48	11.35
Energy efficiency (%)	85.25	79.56	7.59	89.81	82.85	9.51	81.45	73.89	10.96	74.98	64.78	15.63
CASE-3	CHOPPER	PURS	OI (%)	CHOPPER	PURS	OI (%)	CHOPPER	PURS	OI (%)	CHOPPER	PURS	OI (%)
SLA violation rate (%)	24.46	31.64	22.69	29.16	36.56	20.24	42.56	47.91	11.17	48.38	65.61	26.37
No. of missed deadlines	15	18	16.67	28	34	17.65	46	49	6.12	71	78	8.97
CASE-4	CHOPPER	ARCS	OI (%)	CHOPPER	ARCS	OI (%)	CHOPPER	ARCS	OI (%)	CHOPPER	ARCS	OI (%)
Resource contention (sec)	1720.25	2441.48	29.54	3817.56	4280.48	10.82	4980.98	5461.45	8.81	6111.48	6982.78	12.47
CASE-5	CHOPPER	SHAPE	OI (%)	CHOPPER	SHAPE	OI (%)	CHOPPER	SHAPE	OI (%)	CHOPPER	SHAPE	OI (%)
Intrusion detection rate (%)	22.45	18.78	22.22	27.98	26.48	3.85	48.78	44.69	9.09	61.15	55.78	10.91
Fault detection rate (%)	63.15	62.98	1.61	67.48	64.78	4.69	74.98	71.12	4.23	79.48	76.18	3.95
Availability (%)	82.84	80.13	3.37	86.39	82.71	4.46	89.77	89.22	0.62	92.69	90.12	2.89
Reliability (%)	7.33	6.95	5.47	7.91	7.23	9.41	8.42	8.18	3.95	9.12	8.63	4.65
Waiting time (sec)	319.25	323.12	1.24	301.32	308.69	2.27	268.69	276.15	2.90	231.98	248.23	6.85
Turnaround time (sec)	687.98	703.25	2.28	622.25	661.55	5.90	561.89	593.28	5.40	502.36	547.99	8.23

age waiting time by 3.32%, average SLA violation rate by 20.12%, average execution time by 19.29%, average resource contention by 15.41%, average energy consumption by 24.06%, average number of missed deadlines by 12.35%, average turnout time by 5.45% and average execution cost by 16.79% as compared to existing resource management techniques. CHOPPER provides an effective outcome with 16 core processor as compared to other processor configurations; 6.65% better than 8 core processor, 9.34% better than 4 core processor and 15.36% better than 2 core processor. From all the experimental results, it is clearly shown that CHOPPER performs better than existing techniques in terms of QoS parameters because CHOPPER manages every situation automatically.

5 Conclusions and future scope

In this paper, we have presented the self-management properties in the context of autonomic cloud computing. We have presented QoS-aware autonomic resource management approach named CHOPPER. CHOPPER efficiently schedules the provisioned cloud resources automatically and maintains the SLA based on user's QoS requirements to reduce the human intervention and improves user satisfaction. The algorithms for three different phases (monitoring, analyses and plan and execution) of self-management have been developed by focusing on important aspects of self-configuration, self-healing, self-protection and self-optimization. We have examined the effects of various QoS parameters including fault detection rate, throughput, reliability, availability, waiting time, turnaround time, intrusion detection rate, resource utilization, SLA violation rate, execution cost, execution time, energy efficiency and resource contention. Workloads are classified into three categories based on deadline urgency: (i) normal (relaxed deadline), (ii) good (average deadline) and (iii) critical (urgent deadline). Further, we have investigated the impact of different workloads on different QoS parameters. Performance of CHOPPER has been evaluated with existing resource management techniques and improves security, energy efficiency, reliability and availability of cloud based services in real cloud platforms.

Our experimental results provide evidence that the proposed approach can be used to improve scalability of cloud based services. Future research directions for extending the work to support other characteristics of autonomic systems are:

- CHOPPER can be extended further to add sensitivity of assumptions in weight calculations of both homogenous and heterogeneous cloud workloads. Cloud providers can use these results to quickly assess possible reductions in execution time and cost, hence having the potential to save energy.
- CHOPPER can also be extended by identifying relationship between workload (patterns) and the resource demands (demands for compute, storage, and network resources) in the cloud which will further improve the performance.

Acknowledgements One of the authors, Dr. Sukhpal Singh Gill [Post Doctorate Fellow], gratefully acknowledges the CLOUDS Lab, School of Computing and Information Systems, The University of Melbourne, Australia, for awarding him the Fellowship to carry out this research work.

References

1. Singh, S., Chana, I.: QoS-aware autonomic resource management in cloud computing: a systematic review. ACM Comput. Surv. **48**(3), 1–46 (2015)
2. 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, pp. 51–72. Springer International Publishing, Cham (2014)
3. Singh, S., Chana, I.: QoS-aware Autonomic Cloud Computing for ICT. In: The Proceedings of International Conference on Information and Communication Technology for Sustainable Development (ICT4SD-2015), Ahmedabad, India, 3–4 July, 2015. Springer International Publishing, Cham (2015)
4. Singh, S., Chana, I.: Q-aware: Quality of service based cloud resource provisioning. Comput. Electr. Eng. **45**, 138–160 (2015)
5. Singh, S., Chana, I.: QRSF: QoS-aware resource scheduling framework in cloud computing. J. Supercomput. **71**(1), 241–292 (2015)
6. Singh, S., Chana, I.: EARTH: energy-aware autonomic resource scheduling in cloud computing. J. Intell. Fuzzy Syst. **30**(3), 1581–1600 (2016)
7. Broto, L., Hagimont, D., Stolf, P., Depalma, N., Temate, S.: Autonomic management policy specification in tune. In: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 1658–1663. ACM (2008)
8. Valeria, C., Casalicchio, E., Lo Presti, F., Silvestri, L.: Sla-aware resource management for application service providers in the cloud. In: 2011 First International Symposium on Network Cloud Computing and Applications (NCCA), pp. 20–27. IEEE (2011)
9. Mosallanejad, A., Atan, R., Murad, M.A., Abdullah, R.: A hierarchical self-healing SLA for cloud computing. Int. J. Digit. Inf. Wirel. Commun. (IJDIWC) **4**(1), 43–52 (2014)
10. Feller, E., Rilling, L., Morin, C.: Snooze: a scalable and autonomic virtual machine management framework for private clouds. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 482–489. IEEE Computer Society (2012)
11. Malik, S., Huet, F.: Adaptive fault tolerance in real time cloud computing. In: Services (SERVICES), 2011 IEEE World Congress on, pp. 280–287. IEEE (2011)
12. Maurer, M., Brandic, I., Sakellariou, R.: Adaptive resource configuration for cloud infrastructure management. Future Gener. Comput. Syst. **29**(2), 472–487 (2013)
13. 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)

14. You, X., Wan, J., Xianghua, X., Jiang, C., Zhang, W., Zhang, J.: Aras-m: automatic resource allocation strategy based on market mechanism in cloud computing. J. Comput. **6**(7), 1287–1296 (2011)
15. Emeakaroha, V.C., Netto, M.A.S., Calheiros, R.N., Brandic, I., Buyya, R., De Rose, C.A.F.: Towards autonomic detection of SLA violations in Cloud infrastructures. Future Gener. Comput. Syst. **28**(7), 1017–1029 (2012)
16. Bu, X., Rao, J., Cheng-Zhong, X.: Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. IEEE Trans. Parallel Distrib. Syst. **24**(4), 681–690 (2013)
17. Lama, P., Zhou, X.: Aroma: automated resource allocation and configuration of mapreduce environment in the cloud. In: Proceedings of the 9th international conference on Autonomic computing, pp. 63–72. ACM (2012)
18. Kijsipongse, E., Vannarat, S.: Autonomic resource provisioning in rocks clusters using eucalyptus cloud computing. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems, pp. 61–66. ACM (2010)
19. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: 2010 11th IEEE/ACM International Conference on Grid Computing (GRID), pp. 41–48. IEEE (2010)
20. Sah, S.K., Joshi, S.R.: Scalability of efficient and dynamic workload distribution in autonomic cloud computing. In: 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), pp. 12–18. IEEE (2014)
21. Khargharia, B., Hariri, S., Yousif, M.S.: Autonomic power and performance management for computing systems. Clust. Comput. **11**(2), 167–181 (2008)
22. Bashar, A.: Autonomic scaling of Cloud Computing resources using BN-based prediction models. In: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), pp. 200–204. IEEE (2013)
23. Sheikhalishahi, M., Grandinetti, L., Wallace, R.M., Vazquez-Poletti, J.L.: Autonomic resource contention-aware scheduling. Softw: Pract. Exp. **45**(2), 161–175 (2015)
24. Qu, G., Rawashdeh, O.A., Che, D.: Self-protection against attacks in an autonomic computing environment. IJ Comput. Appl. **17**(4), 250–256 (2010)
25. 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)
26. Chopra, I., Singh, M.: SHAPE–an approach for self-healing and self-protection in complex distributed networks. J. Supercomput. **67**(2), 585–613 (2014)
27. Kephart, J.O., Walsh, W.E.: An architectural blueprint for autonomic computing. Technical Report, IBM Corporation (2003), 1–29, IBM. Retrieved on December 25, 2014 from: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
28. Broto, L., Stolf, P., Bahsoun, J.-P., Hagimont, D., Depalma, N.: Specifying self-administered policies with Tune. In: French Conference on Operating Systems (CFSE). Fribourg (2008)
29. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (IoT): a vision, architectural elements, and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
30. Chu, X., Nadiminti, K., Jin, C., Venugopal, S., Buyya, R.: Aneka: next-generation enterprise grid platform for e-science and e-business applications. In: Proceeding of the IEEE International Conference on e-Science and Grid Computing, pp. 151–159. IEEE (2007)
31. 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)
32. Singh, S., Chana, I.: Efficient Cloud Workload Management Framework. Masters Dissertation, Thapar University, Punjab (2013)
33. Singh, S., Chana, I.: Resource provisioning and scheduling in clouds: QoS perspective. J. Supercomput. **72**(3), 926–960 (2016)
34. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Ming, X.: Web services agreement specification (WS-Agreement). In Open Grid Forum **128**, 216 (2007)
35. Simão, J., Veiga, L.: Partial utility-driven scheduling for flexible SLA and pricing arbitration in clouds. IEEE Trans. Cloud Comput. **4**(4), 467–480 (2016)
36. Singh, S., Chana, I., Singh, M., Buyya, R.: SOCCER: self-optimization of energy-efficient cloud resources. Clust. Comput. **19**(4), 1787–1800 (2016)
37. Singh, S., Chana, I., Buyya, R.: STAR: SLA-aware autonomic management of cloud resources. In: IEEE Transactions on Cloud Computing, pp.1-14, doi:10.1109/TCC.2017.2648788, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7807337&isnumber=6562694
38. Sharrock, R., Monteil, T., Stolf, P., Hagimont, D., Broto, L.: Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements. Int. J. Adapt. Resil. Auton. Syst. IGI Glob. Hershey—USA **2**(2), 58–76 (2011)
39. Hagimont, D., Stolf, P., Broto, L., Depalma, N.: Component-based autonomic management for legacy software. In: Denko, M., Yang, L., Zhang, Y. (eds.) Autonomic Computing and Networking, pp. 83–104. Springer, New York (2009). 978-0-387-89827-8
40. Toure, M., Berhe, G., Stolf, P., Broto, L., Depalma, N., Hagimont, D.: Autonomic management for grid applications. In: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), pp. 79–86. IEEE (2008)
41. Singh, S., Chana, I., Singh, M.: The journey of QoS based autonomic cloud computing. IT Prof. Mag. **19**(2), 42–49 (2017)
42. Chetsa, G.L.T., Lefèvre, L., Pierson, J.-M., Stolf, P., Da Costa, G.: Exploiting performance counters to predict and improve energy performance of HPC systems. Future Gener. Comput. Syst. **36**, 287–298 (2014)

Sukhpal Singh Gill joined Computer Science and Engineering Department of Thapar University, Patiala, India, in 2016 as a Faculty. Dr. Gill obtained the Degree of Master of Engineering in Software Engineering from Thapar University, as well as a Doctoral Degree specialization in “Autonomic Cloud Computing” from Thapar University. Presently, Dr. Gill is working as Post Doctorate Fellow at CLOUDS Lab, School of Computing and Information Systems, The University of Melbourne, Australia. Dr. Gill received the Gold Medal in Master of Engineering in Software Engineering. Dr. Gill is a DST Inspire Fellow [2013–2016] and worked as a SRF-Professional on DST Project, Government of India. He has done certifications in Cloud Computing Fundamentals, including Introduction to Cloud Computing and Aneka Platform (US Patented) by ManjraSoft Pty Ltd, Australia and Certification of Rational Software Architect (RSA) by IBM India. His research interests include Software Engineering, Cloud Computing, Internet of Things and Fog Computing. He has more than 40 research publications in reputed journals and conferences.

Inderveer Chana joined Computer Science and Engineering Department of Thapar University, Patiala, India, in 1997 as Lecturer and is presently serving as Professor in the department. She is Ph.D. in Computer Science with specialization in Grid Computing, M.E. in Software Engineering from Thapar University and B.E. in Computer Science and Engineering. Her research interests include Grid and Cloud computing and other areas of interest are Software Engineering and Software Project Management. She has more than 100 research publications in reputed Journals and Conferences. Under her supervision, more than 40 ME thesis and seven Ph.D thesis have been awarded and five Ph.D. thesis are on-going. She is also working on various research projects funded by Government of India.

Maninder Singh received his Bachelor's Degree from Pune University in 1994, and holds a Master's Degree, with honors in Software Engineering from Thapar Institute of Engineering & Technology, as well as a Doctoral Degree specialization in Network Security from Thapar University. Dr. Singh is currently working as Associate Professor in Computer Science and Engineering Department at Thapar University. Dr. Singh is on the Roll-of-honour at EC-Council USA, being certified as Ethical Hacker (C—EH), Security Analyst (ECSA) and Licensed Penetration Tester (LPT). Dr. Singh has successfully completed many consultancy projects for renowned national bank(s).

Rajkumar Buyya is a Fellow of IEEE, Professor of Computer Science and Software Engineering and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft, a spin-off company of the University, commercialising its innovations in Cloud Computing. He has authored over 500 publications and four text books. He is one of the highly cited authors in computer science and software engineering worldwide (h-index 110+, 61,000+ citations). He has served as the founding Editor-in-Chief (EiC) of IEEE Transactions on Cloud Computing and now serving as Co-EiC of Journal of Software: Practice and Experience.