This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

STAR: SLA-aware Autonomic Management of Cloud Resources

Sukhpal Singh, Inderveer Chana and Rajkumar Buyya, Fellow, IEEE

Abstract
Cloud computing has recently emerged as an important service to manage applications efficiently over the Internet. Various cloud providers offer pay per use cloud services that requires Quality of Service (QoS) management to efficiently monitor and measure the delivered services through Internet of Things (IoT) and thus needs to follow Service Level Agreements (SLAs). However, providing dedicated cloud services that ensure user's dynamic QoS requirements by avoiding SLA violations is a big challenge in cloud computing. As dynamism, heterogeneity and complexity of cloud environment is increasing rapidly, it makes cloud systems insecure and unmanageable. To overcome these problems, cloud systems require self-management of services. Therefore, there is a need to develop a resource management technique that automatically manages QoS requirements of cloud users thus helping the cloud providers in achieving the SLAs and avoiding SLA violations. In this paper, we present SLA-aware autonomic resource management technique called STAR which mainly focuses on reducing SLA violation rate for the efficient delivery of cloud services. The performance of the proposed technique has been evaluated through cloud environment. The experimental results demonstrate that STAR is efficient in reducing SLA violation rate and in optimizing other QoS parameters which effect efficient cloud service delivery.

Index Terms: Autonomic Cloud, Resource Provisioning, Cloud Computing, Resource Scheduling, Quality of Service, Service Level Agreement

1. Introduction

CLOUDS offer three types of services such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) and therefore it requires management of Quality of Service (QoS) to efficiently monitor and measure the delivered services to meet Service Level Agreements (SLAs). In Cloud environment, uncertainty and dispersion of resources encounters problems in efficient management of resources, which is caused due to many reasons [1] [2] such as: i) heterogeneity (due to different type of resources and scheduling techniques), ii) dynamism (detect and fulfill the requirements of application at runtime) and iii) failures (failure of system or resources which leads to performance degradation). However, present cloud computing systems and management techniques are unable to handle above mentioned problems efficiently at runtime. An autonomic system provides a solution to this problem by offering the environment in which applications can be managed efficiently by fulfilling QoS requirements of applications without human involvement. Thus, autonomic cloud system becomes self-managed to overcome the above challenges and to provide reliable, secure and cost efficient services to end users.

Currently, cloud services are provisioned and scheduled according to resources' availability without ensuring the expected performances [3]. The cloud provider should evolve its ecosystem in order to meet QoS requirements of each cloud component. To realize this, there is a need to consider two important aspects which reflect the complexity introduced by the cloud management should be considered: firstly QoS-aware and secondly autonomic management of cloud services.

- *S. Singh and I. Chana is with the Computer Science and Engineering Department, Thapar University, Patiala, Punjab, India-147004. E-mail: {ssgill,inderveer}@thapar.edu and @thapar.edu.*
- *R. Buyya is with the CLOUDS Lab, Department of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3010, Australia. E-mail: rbuyya@unimelb.edu.au.*

QoS-aware aspect involves the capacity of a service to be aware of its behavior to ensure the elasticity, high availability, reliability of service, cost, time etc. as mentioned in SLA [4]. Autonomic implies the fact that the service is able to self-manage itself as per its environment needs. Thus, maximizing cost-effectiveness and resource utilization for applications while ensuring performance and other QoS guarantees, requires leveraging important and extremely challenging tradeoffs [5].

Based on policy guidance, autonomic system keep the system stable in unpredictable conditions and adapt quickly in new environmental conditions like software, hardware failures etc. Thus, there is a need of SLA-aware autonomic resource management technique which considers all the important QoS parameters like availability, cost, latency, execution time etc. to reduce SLA violation rate for better resource management. In our earlier work [1] [2] [8] [9], we have identified various research issues related to QoS and SLA for autonomic management of cloud resources [1] and based on these issues, we have developed a QoS based resource provisioning technique (Q-aware) to map the resources to the workloads based on user requirements described in the form of SLA [8]. Further, resource scheduling framework (QRSF) has been proposed, in which provisioned resources have been scheduled by using different resource scheduling policies (cost, time, cost-time and bargaining based) [9]. In QRSF, manual resource scheduling is considered which further needs lot of human work every time to schedule resources to execute workloads by fulfilling their QoS requirements. The concept of QRSF has been further extended by proposing energy-aware autonomic resource scheduling technique (SOCCER) [2], in which IBM's autonomic computing concept 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. Our existing research work considers only few QoS parameters of self-optimizing without considering SLA violation rate. This research work is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

an extension of our previous research work [1]. In this research work, we have presented cloud based SLA-aware autonomic resource management technique for both homogenous and heterogeneous cloud workloads and measured the impact of QoS parameters on SLA violation rate.

The primary aim of this paper is to develop a SLA-aware autonomic cloud resource management technique called **STAR** *(**S**LA-aware autonomic **T**echnique for **A**llocation of **R**esources)* for effective scheduling of resources which considers SLA violation rate along with other QoS parameters like execution time, cost, latency, reliability and availability. The objectives of this research work are: i) to propose an autonomic resource management technique for execution of heterogeneous workloads by considering generic property of self-management, ii) to optimize the above mentioned QoS parameters, iii) to reduce SLA violation rate and improve user satisfaction by fulfilling their QoS requirements and iv) to implement and perform evaluation in cloud environment.

The rest of the paper is organized as follows: Section 2 presents the related work. Proposed technique is presented in Section 3. Sections 4 describe the experimental setup and present the results of evaluation. Section 5 presents conclusion and future directions.

2. Related Work

Several researchers have investigated the area of SLA-aware resource management in Cloud computing. Rajkumar et al. [3] proposed SLA-oriented resource management mechanism which provisions combination of virtualization technologies and market based resource management policies for flexible resource allocation to applications. By following dynamic allocation of resources, this mechanism is efficient in fulfilling QoS requirements of applications. Further, system efficiency can be improved by incorporating the concept of autonomic resource management. Hossein and Mohammad [4] proposed a Proactive Resource Allocation (PRA) model to reduce the impact of SLA violations which considered customer satisfaction level as a significant element in profitability for cloud providers. This model aids provider to recognize particular actions that can improve preservation and cost-effectiveness in the long run. Jose and Luıs [5] 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. Artur et al. [6] proposed probabilistic method for the optimization of reliability, performance and monetary costs, given application and customer requirements and dynamic constraints. Further, performance is evaluated using real instance price traces and workload models to validate its effectiveness. Vatche et al. [7] proposed a framework for workload colocation called CloudPack which offers consumers with the capability to legally define Directed Acyclic Graphs based workload flexibilities, improves the use of resources to reduce total costs.

Andrés et al. [10] proposed SLA-driven dynamic cloud resource management called Cloudcompaas which allows providers with a general SLA model to manage higher-level metrics, nearer to user's perception, and with flexible structure of the requirements of many users. Under highly heterogeneous utilization patterns, it attains least cost and extreme efficiency. Detecting SLA Violation infrastructure (DeSVi) [11] uses resource monitoring mechanism to prevent the violation of SLA. DeSVi allocates the resources the workloads in virtual environment and resources are monitored by mapping user defined SLA with low-level resource metrics. Service level objectives have been defined to detect the violation in SLA and resource utilization. DeSVi executes transactional web applications and image rendering applications which contains heterogeneous workloads and monitors consumption of resources. Damián et al. [12] proposed SLA aware Service (SLAaaS) which considers SLA and QoS levels as first preference of cloud users and proved that SLAaaS is successfully managing SLA with diverse QoS requirements of cloud services such as financial energetic costs, dependability and performance. Saurabh et al. [13] proposed a scheduling and an admission control method which not only improves the CPU utilization and revenue, but also guarantees that the customer's QoS requirements are fulfilled as stated in SLAs. Further, it offers considerable enhancement over static server consolidation and decreases SLA violations. Emiliano and Luca [14] analyzed the application service provider viewpoint's problem which uses cloud resources to attain scalable provisioning of its services in terms of QoS requirements. Zhou et al. [15] proposed adaptive threshold energy-aware algorithm in order to decrease the SLA violation and energy consumption. Algorithm transfers virtual machines on little or heavily loaded hosts to lightly loaded hosts, while the virtual machines on lightly and reasonably loaded hosts remain unaffected. Seokho and Sung [16] proposed a SLA negotiation method for flexible and interactive SLA creation and it provision innovative multi-issue negotiation which contains price and time slot negotiations. This algorithm is effective in terms of provider's profits and SLA violations. Linlin et al. [17] proposed automated negotiation technique where a Software-as-a-Service (SaaS) broker is used as the one-stop-shop for consumers to attain the necessary service proficiently when compromising with many providers. Further, counter offer generation strategies are designed to improve user satisfaction levels and profit for the broker. Linlin et al. [18] proposed consumer oriented SLA-based resource management mechanism to reduce cost by reducing penalty cost and increase customer satisfaction level by reducing SLA violations. This mechanism considers providers' quality parameters and customer profiles to manage dynamic user requests and infrastructure level heterogeneity. Hadi and Massoud [19] et al. hierarchical SLA based resource scheduling technique which considers the energy non-proportionality of current servers, cooling power consumption and peak power checks. Proposed solution is effective in reducing the operational cost while fulfilling SLA constraints and reduces execution time. Seokho et al. [20] proposed SLA-based resource management mechanism that takes into account the geographical location of distributed data centers and workloads. Further, experimental results show that it performs better than round robin approach, greedy approach, and manual allocation of resources in terms of provider's profits and SLA violations. Linlin at al. [21] proposed SLA-based Resource Allocation (SRA) mechanism to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

map the user workloads to available resource, fulfill QoS requirements of user application at runtime and implemented in virtual environment. Further, SRA considers QoS parameters like response time and service time and cost and SLA violations. But SRA failed to analyze the effect of QoS parameters on SLA violation rate.

Although the above research works have presented SLA-aware resource management techniques in cloud computing by focusing on different QoS parameters, they have not investigated the impact of QoS parameters on SLA violation rate. Due to this, the current resource management services may not be able to deliver very efficient services and may need to compromise SLA violation. Our proposed SLA-aware autonomic resource management technique *(STAR)* considers possible QoS parameters of SLA and measures the impact of QoS parameters on SLA violation rate and thus addresses the deficiency in the state-of-the-art.

3. STAR Architecture

The architecture of *STAR (SLA-aware autonomic Technique for Allocation of Resources)* system, which measures the impact of QoS parameters on SLA violation rate, is shown in Figure 1. STAR is the key mechanism that ensures that resource manager can serve large amount of requests without violating SLA agreement and dynamically manages the resources based on QoS requirements identified by QoS manager.

3.1 QoS based Metrics

The following metrics ([Eq. 1] – [Eq. 6]) are selected from our previous work [2] [8] [9] to calculate the SLA violation rate, execution time, SLA cost, deadline time, workload deadline and average execution cost. The goal of cloud provider is to minimize the SLA violation rate. The cloud workload will be executed only when the SLA violation rate is less than the threshold value of SLA violation rate (maximum SLA deviation agreed between cloud provider and user).

SLA Violation Rate is the *product* of *Failure Rate* and *Weight of SLA* and can be calculated as [Eq. 1]. List of SLA = $<m_1, m_2 \ldots\ldots\ldots\ldots m_n>$, where n is total number of SLAs

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

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

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

Where w_i is weight for every SLA. [Eq. 2] is used to calculate Execution Time (Et_i).

$$\text{Execution Time}\ (Et_i) = \sum_{i=1}^{n} \left(\frac{WC_i - WS_i}{n} \right) + \Delta t_i \qquad (2)$$

Where WC_i is workload completion time and WS_i is workload submission time, Δt_i is time to restart the node and n is the number of workloads. Based on SLA Violation Rate (SVR), SLA Cost (SC_i) is calculated using following formula [Eq. 3]:

$$\text{SLA Cost}\ (SC_i) = SVR \times \sum_{i=1}^{n} (Et_i) \qquad (3)$$

Where $\boldsymbol{i}$ is number of workloads. The range of SVR is decided based on the margin by which deadline of workload is missed (See Table 4). [Eq. 4] is used to calculate *Deadline Time*.

$$Deadline\ Time\ (Dt_i) = \sum_{i=1}^{n} (Wd_i - Ct_i) \qquad (4)$$

Where Wd_i is workload deadline and Ct_i is current time. *Workload deadline* (Wd_i) is used to calculate the final priority of workload [Eq. 5].

$$\text{Workload deadline}\ (Wd_i) = \sum_{i=1}^{n} \left(\frac{Dd_i}{Et_i} - 1 \right) \qquad (5)$$

Where Dd_i is desired deadline and ET_i is execution time calculated using [Eq. 2]. *Execution Cost* is an addition of resource cost and penalty cost. STAR defined the different levels of penalty rate based on QoS requirements. Delay time is difference of deadline and time when workload is actually completed. We have used following formula to calculate Execution Cost (C) [Eq. 6].

$$C = Resource\ Cost + Penalty\ Cost \qquad (6)$$

$$Resource\ Cost = Et_i \times Price$$

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

$$Delay\ Time = Expected\ Completion\ Time - Actual\ Completion\ Time$$

$$PC = \begin{cases} Penalty_{minimum}, & if\ Expected\ Completion\ Time \geq Actual\ Completion\ Time \\ Penalty_{minimum} + [Penalty\ Rate \times |Delay\ Time|], & if\ Expected\ Completion\ Time < Actual\ Completion\ Time \end{cases}$$

Where $c \in C$, C is set of penalty cost with different levels specified in STAR. The assumptions of proposed SLA-aware autonomic resource management technique are: a) Multi user accessing the cloud based system simultaneously, b) Workloads have different execution time and different deadlines and c) Cloud users have different QoS parameters and can be changed dynamically. The units of proposed SLA-aware autonomic resource management technique are described below: First of all cloud consumer tries to execute the workloads through the *Cloud Workload Management Portal* (CWMP) i.e. web based application. After authentication, STAR 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. STAR 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 to each other using CWMP and update their new rules and polices of resources on cloud in the STAR as shown in Figure 1. 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. 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 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

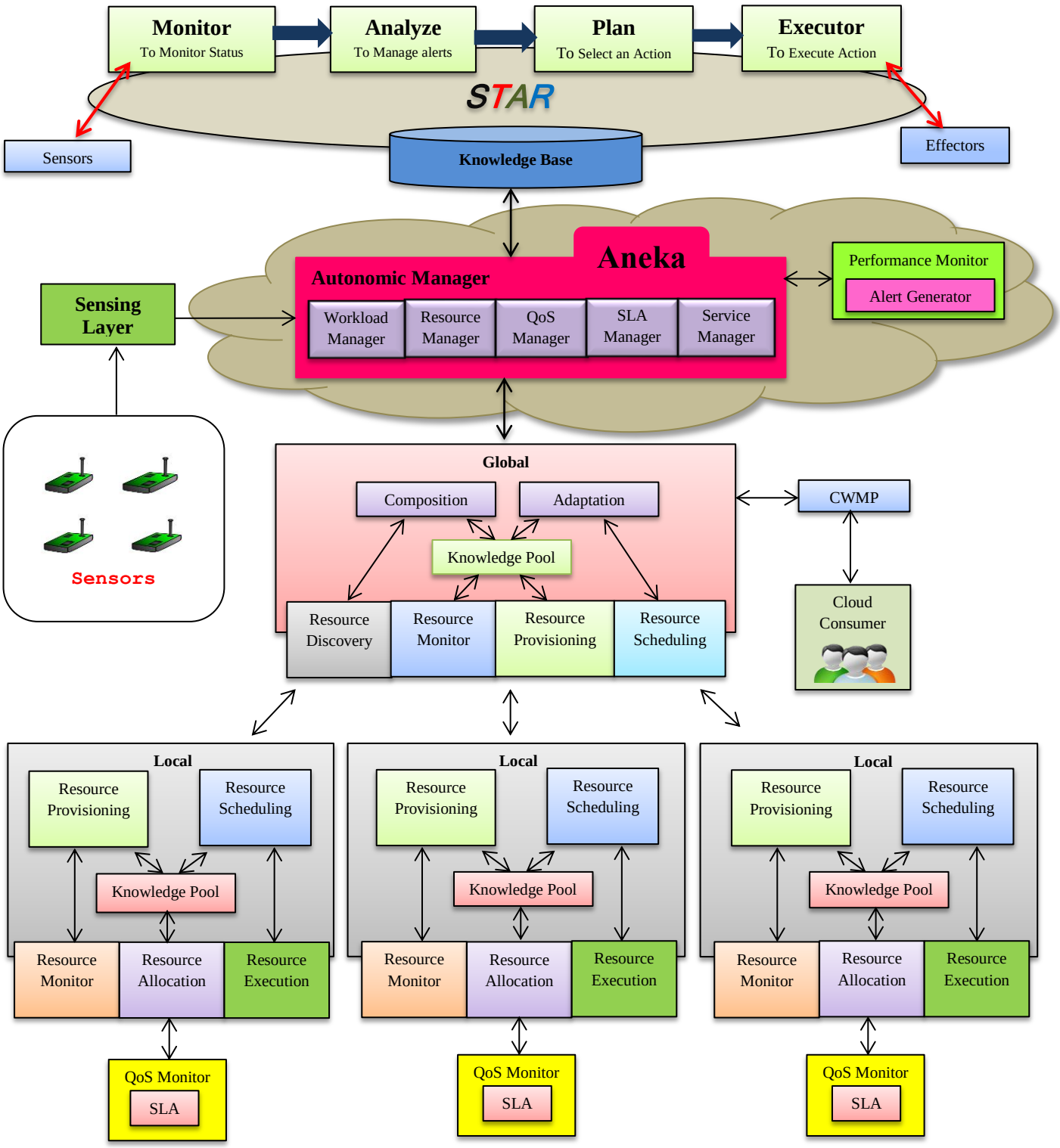

Fig. 1: STAR Architecture

The types of workload that have been considered for this research work are: websites, technological computing, endeavor software, performance testing, online transaction processing, e-com, central financial services, storage and backup services, production applications, software/project development and testing, graphics oriented, critical internet applications and mobile computing services [8] [9]. 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 before actual execution. Finally, workload generates output in the form of workload information.

Resource management in STAR is done at two levels: Global and Local as shown in Figure 1. In Global level, cloud consumer interacts with system to submit their workload or application for execution along with SLA pertaining QoS requirements. Task of execution of workload or application is divided into sub tasks or small levels called local levels. The actual execution of workload or application is performed at local level after verification of availability of resources. The *knowledge pool* stores the predefined rules defined by system administrator and rules will be updated time to time based on new polices of resource allocation. Aim of *Resource allocation* is to allocate appropriate resources to the suitable workloads on time, so that applications can utilize the resources effectively. SLA-aware autonomic resource management is done based on IBM's autonomic

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

model [23] that considers four steps of autonomic system: 1) Monitor, 2) Analyze, 3) Plan and 4) Execute, in which Autonomic Element (AE) is an agent which accepts the input and produces final results based on QoS parameters defined in SLA. AE interacts with environment through manageability interfaces (sensors and effectors) and takes action according to the input received from sensors and rules defined in knowledge base in low level language. Sensors perform two functions: resource discovery and resource monitoring.

Resource discovery is able to find the adequate (with minimum cost and execution time) resource for workload or application based on QoS requirements through *Composition*. *Composition* is able to determine the best resource workload pair for execution. *Resource monitor* monitors whether the conditions are being fulfilled as specified in policy and ensures that all the resources are provided. All the inputs received from monitors are analyzed and action is taken according to the alert generated by *QoS monitor*. *QoS monitor* is used to verify whether all the QoS attributes defined in SLA are fulfilled or not by using *Adaptation*. If not, then *QoS monitor* generates an alert to provide more resources to fulfill the current demand of application. *Adaptation* function is able to maintain the effective execution in case of sudden change in QoS conditions. Based on QoS requirements and policy of system, resources are provisioned to workload or application and resource provisioning information is sent to user for verification. After successful verification by user, the AE allocates resources to workload(s) for resource scheduling. *Resource executor* performs the final step of resource execution and completes the execution within specified deadline and system continues for other workloads or applications.

QoS manager 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 queue (workload with urgent deadline in submission state and deadline is calculated using [Eq. 4]) and non-critical queue (workload without urgent deadline in submission state) through QoS assessment. For QoS assessment, *QoS manager* calculates the execution time of workload and finds the approximate workload completion time. *Resource manager* maintains resource details which includes the number of CPU using, 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 reserve resources. It contains the information about the available resources and reserved resource along with resource description (resource name, resource type, configuration, availability information, usage information and price of resource) as provided by cloud provider. Based on SLA information (Agreed Service Level Agreement), *SLA manager* prepares SLA document which contains information about SLA Violation Rate (maximum and minimum violation and penalty rate in case of SLA violation) and accordingly urgent cloud workloads would be placed in priority queue (workload with urgent deadline in execution state) for earlier execution. SLA Violation Rate is used to measure the deviation of QoS from predictable with their possible resolution. In case of urgent workloads, if the SLA Violation Rate is more than the threshold value then allocate the reserve resources to the particular workload. We have selected the "Web Services Agreement Specification (WS-Agreement)" standard [23] 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.

Service manager manages the whole service of system in a controlled manner. Based on SLA information, QoS information, workload information and resource information, the resource workload mapper maps the workloads to the appropriate resource by taking care of both SLA and QoS. Resource scheduler is further used to schedule the workloads after mapping of the workloads with available resources based on the policy defined by user and generates the workload schedule [Figure 9 and 10] based on the workload details specified by the user and billing for that execution. Resource scheduler uses minimum number of resources to execute the workloads within specified budget and deadline.

3.2. Autonomic Manager

Based on SLA information, resource information, workload information and QoS information resources are provisioned by Q-aware resource provisioning technique for workload execution [8]. After provisioning of resources, actual resource scheduling is done based on QRSF resource scheduling technique [9]. 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 STAR that generates alert in case of performance degradation. Alerts can be generated by *Alert Generator* 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 by providing compensation). The interaction of cloud user and cloud provider to negotiate SLA [1] is shown in Figure 2. Working of sub units is described in Figure 3 as: Monitor [M], Analyze and Plan [AP] and Executor [E]. The same action is performed twice, if AE fails to correct it then system is treated as down. STAR considers four steps of autonomic system: i) monitor, ii) analyze, iii) plan and iv) execute.

3.2.1 Sensors

Sensors get information about the performance of current state of nodes using in the STAR. *Zig Bee Protocol* is used to tap the data sensed and gathered by various sensors. Firstly, the updated information from processing nodes is transferred to manager node then manager node transfers this information to sensors. Updated information includes information about QoS parameters (execution time, execution cost, availability, reliability, latency and SLA violation rate) of all the systems working under cloud environment and update the information time to time.

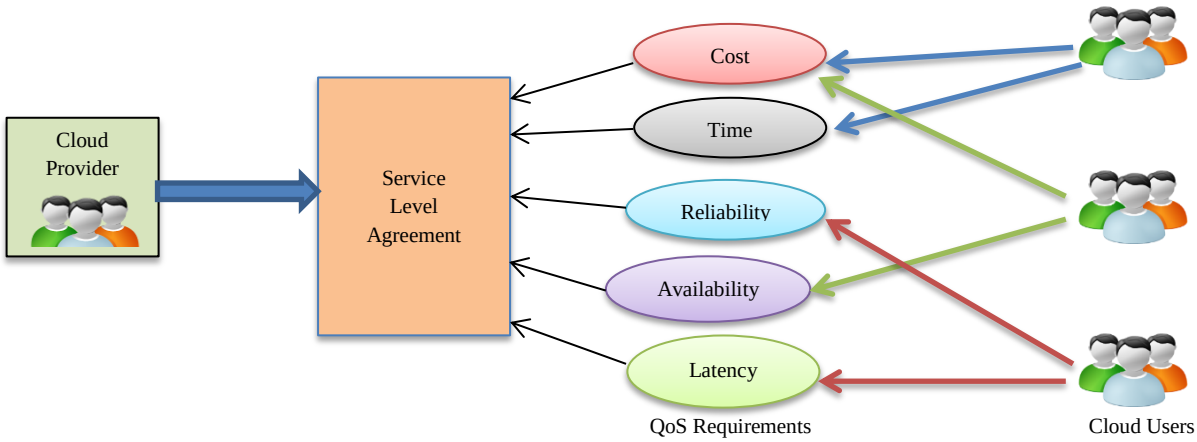

Fig. 2: Process of SLA Negotiation in STAR

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

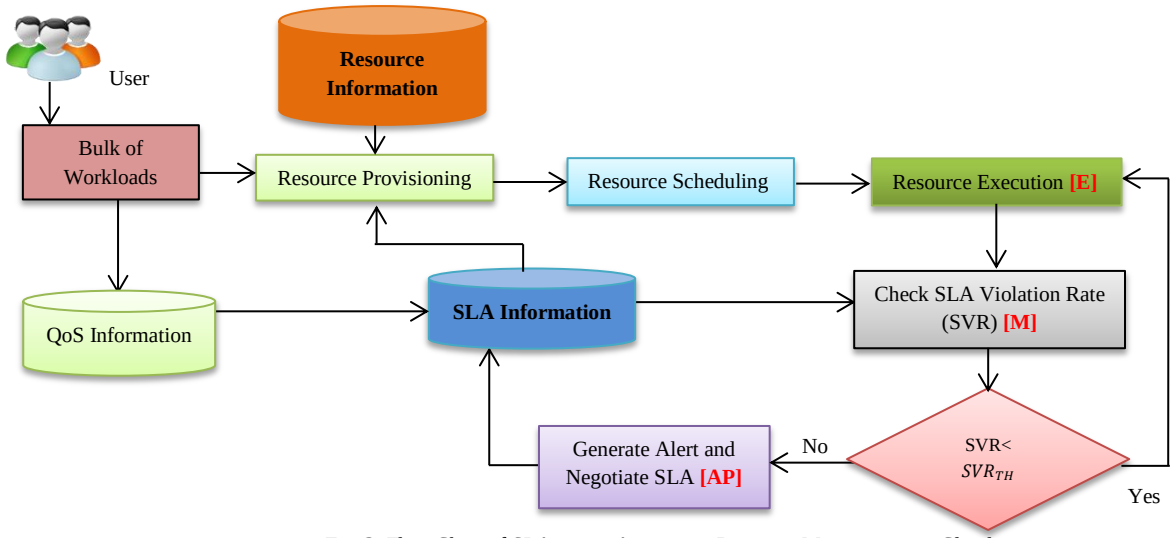

Fig. 3: Flow Chart of SLA-aware Autonomic Resource Management in Cloud

3.2.2 Monitor [M]

Initially, Monitors are used to collect the information from sensors for monitoring continuously the value of QoS parameters and transfer this information to next module for further analysis. [Algorithm 1: Monitoring Unit (MU)] is used to monitor the performance of management of resources by considering SLA Violation Rate is shown in Figure 4. For monitoring SLA Status, QoS agent is installed on all processing nodes to monitor the performance. We have considered set of workloads ($W_Q = \{ W_1, W_2, \ldots\ldots\ldots\ldots\ldots\ldots W_m\}$) placed in workload queue and have considered some or all the workloads for execution based on the availability of resources and QoS requirements of workloads.

ALGORITHM 1: Monitoring Unit (MU)

```
1.  # AUTONOMIC MONITORING
2.  Start
3.  Workload Queue: W_Q = { W_1, W_2, ..... W_m}
4.  Add Workloads: W_a = { W_1, W_2, ..... W_o} where o ≤ m
5.  Allocate resources to workloads based on QoS requirements
6.  for all workloads (W_a), Calculate SLA Violation Rate for execution
7.    if ([SVR < SVR_TH] == 'TRUE') then
8.        Continue execution
9.    else
10.   Generate alert
11.   end if
12. end for
```

Fig. 4: ALGORITHM 1: Monitoring Unit (MU)

After this, resources are allocated to the workloads then SLA Violation Rate (SVR) for every workload will be calculated using [Eq. 1]. If SVR is more than Threshold Value (SVR_{TH}) then alert will be generated otherwise will be continued execution of resources.

3.2.3 Analyze and Plan [AP]

Analyze and plan module starts analyzing the information received from monitoring module and makes a plan for adequate actions for corresponding alert. [ALGORITHM 2: Analyzing Unit (AU)] is used to analyze the performance of management of resources by considering SVR shown in Figure 5. For analyzing SLA Status, 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].

ALGORITHM 2: Analyzing Unit (AU)

```
1.  # AUTONOMIC ANALYZING
2.  # Process logs
3.  # Check for Status of QoS parameters
4.  for all node [Node_current]
5.    if (SVR < SVR_TH) == 'FALSE'
6.    do
7.      Set status Node_current = Down
8.      Add Δt (time to restart) into Execution Time [Eq. 2]
9.      Restart the Node [Node_current ]
10.     if Node_current == 'RESTARTED' then
11.       Check Node status
12.       if Node status [Node_current] = ! 'ACTIVE'
13.         Generate Alert [Node is declared as Dead]
14.       end if
15.     end if
16.   end if
17. end for
```

Figure 5: ALGORITHM 2: Analyzing Unit (AU)

3.2.4 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 self-management property as shown in Figure 6.

ALGORITHM 3: Executing Unit (EU)

```
1.  # AUTONOMIC EXECUTION
2.  for all node [Node_current]
3.    if (SVR < SVR_TH) == 'FALSE' then
4.      Declared node as dead node and removed
5.    else
6.    # Check Resource Availability
7.      if (Resource Available == Yes) then
8.        Add new node (SVR < SVR_TH)
9.      else
          Add new node from reserve resource pool
10.     end if
11.   end if
12. end for
```

Fig. 6: ALGORITHM 3: Executing Unit (AU)

For executing the plan, main goal of executor is to optimize the performance of QoS parameters and execute the workloads without violation of SLA. Based on the information provided by analyzer, executor will add new node from resource pool with minimum SVR. If the resources are not available in resource pool then add new node from reserve resource pool with minimum SVR. During execution of workloads, performance is monitored continuously using a sub unit *performance monitor* to maintain the efficiency of system and generates alert in case of performance degradation as shown in Figure 1. Alerts can be generated if the SVR is more than Threshold Value of SVR.

3.2.5 Effector

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

4. Performance Evaluation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

Tools used for setting up cloud environment are Microsoft Visual Studio 2010 [SaaS], Aneka [PaaS] and SQL Server 2008, JADE Platform (for agents) and Citrix Xen Server [IaaS]. The integration of multiple environments used to conduct experiment is shown in Figure 7.

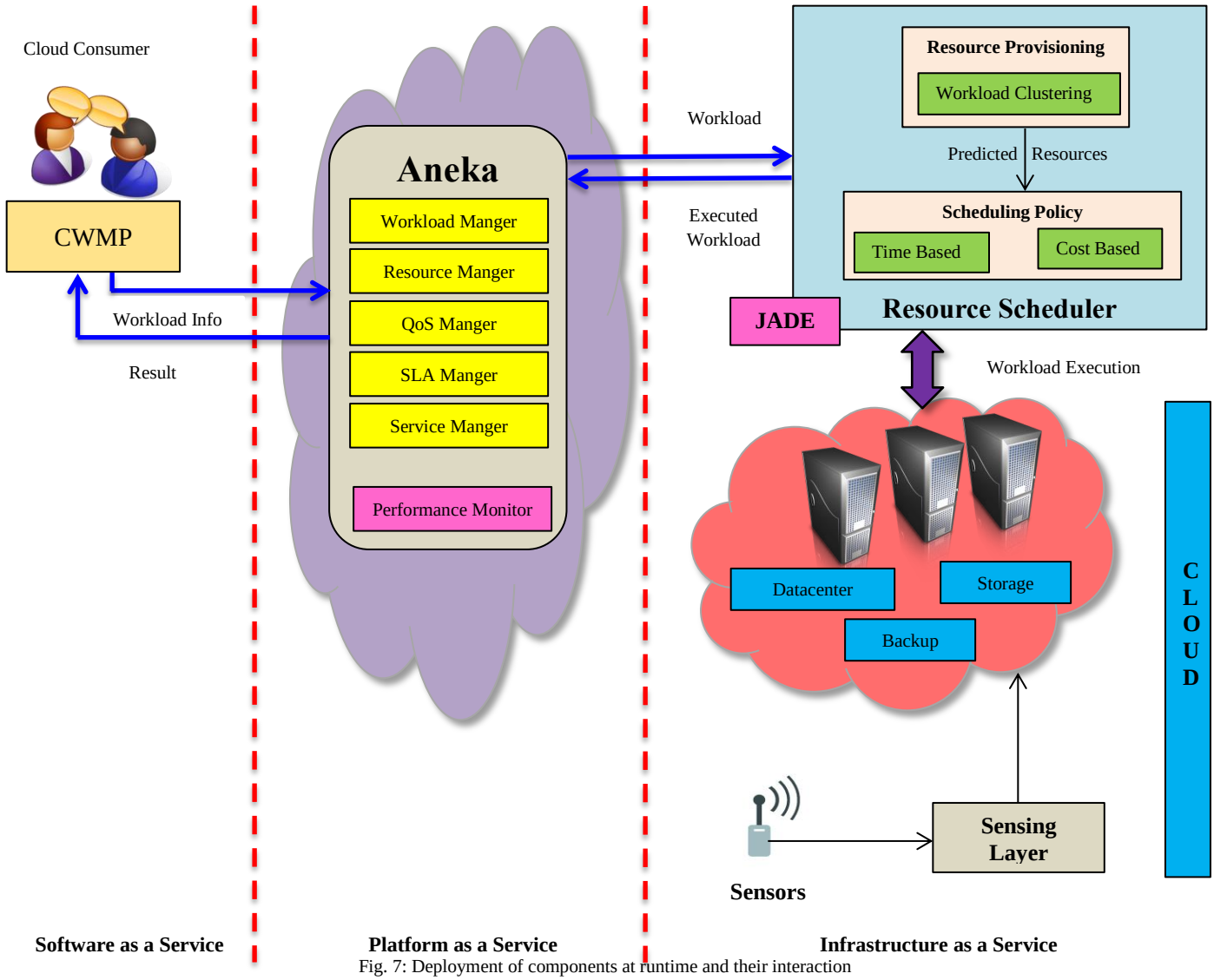

Fig. 7: Deployment of components at runtime and their interaction

TABLE 1
CONFIGURATION DETAILS OF THAPAR CLOUD

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

STAR is installed on main server and tested on virtual cloud environment that has been established at *High Performance Computing Lab at Thapar University, India*. We have installed different number of virtual machines on different servers, and deployed the SLA-aware autonomic resource management technique 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 SaaS level, Microsoft Visual Studio is used to develop Cloud Workload Management Portal (CWMP) to provide user interface in which user can access service from any geographical location. At PaaS level, Aneka cloud application platform [22] is used as a scalable cloud middleware to make interaction between IaaS and SaaS, and continually monitor the performance of the system. Aneka task model has been used in this research work. A task is a single unit of work processed (workload) in a node. It is independent from other tasks that may be executed on the same or any other node at the same time.

In STAR, Aneka task model involves the following components: Workload Manager, QoS Manager, Resource Manager, SLA Manager, Service Manager and Performance Monitor. At ***IaaS level***, three different servers (consist of virtual nodes) have been created through Citrix Xen Server and SQL Server has been used for data storage. Computing nodes used in this experiment work are further categorized into three categories as shown in Table 1.

The execution cost is calculated based on user workload and deadline (if deadline is too early (urgent) then it will be more costly because we need a greater processing speed and free resources to process particular workload with urgency). Their individual price is fixed (artificially) for different resources because all the resources are working in coordination manner to fulfill the demand of user (demand of user is changing dynamically). Experiment setup using 3 servers (R1, R2 and R3) in which further virtual nodes (12 = 6 (Server 1) +4 (Server 2) +2 (Server 3)) are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

created. Every virtual node has different number for Execution Components (ECs) to process user workload and every EC has their own cost (C$/EC time unit (Sec)). Table 1 shows the characteristics of the resources used and their Execution Component (EC) access cost per time unit in Cloud dollars (C$) 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. Due to limited number of resources, cost increases with increase in user workloads. Cost is varying in two different cases: i) relaxed deadline and ii) tight deadline. In both cases, when the deadline is low (e.g. 200 secs), the number of user workloads processed increases as the budget value increases. When a higher budget is available, the execution agent uses expensive resources to process more user workloads within the deadline. Alternatively, when scheduling with a low budget, the number of user workloads processed increases as the deadline is relaxed. Execution agent allocates as many user workloads as the first cheapest resource can complete by the deadline, and then allocates the remaining user workloads to the next cheapest resources. When the deadline is tight (e.g. 100), there is high demand for all the resources in a short time. All the resources are used up so long as budget is available to process all user workloads within the deadline. However, when the deadline is relaxed (e.g. 700 secs), it is likely that all user workloads can be completed using the first few cheapest resources. As the deadline increases, execution agent schedules user workloads on the available resources to finish earlier as possible. After studying and confirming the various QoS constraints which the workload has required, Resource Provisioning Unit (RPU) checks the availability of resources. The different cloud workloads have different set of QoS requirements and characteristics. All the workloads are submitted to RPU are analyzed based on their QoS requirements. For QoS, the required workload patterns are identified for clustering of workloads then identifies the QoS metrics required to assign the weights based on level of measurement described in QoS requirements specified in SLA.

TABLE 2
CLOUD WORKLOADS, WORKLOAD TYPE AND THEIR QOS REQUIREMENTS

Workload Name	QoS Requirements	Workload Type
Web sites	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	SLA Cost and Execution Time	Compute
Online Transaction Processing	Security, High availability, Internet accessibility, Usability	Administration
E-Com	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

Further, *K-means* based clustering algorithm is used for re-clustering the workloads for execution on different set of resources [8]. RPU provisions the required resources to the clustered workload for their execution in cloud environment. After resource provisioning, the role of resource scheduler starts. Scheduler as shown in Figure 7, runs at IaaS level on Citrix Xen Server. For efficient scheduling and optimum resource usage, the scheduler works on the basis of two different resource scheduling policies (time based scheduling policy and cost based scheduling policy)] schedules the resources for execution of workloads [9]. In *time based scheduling policy*, QoS manager calculates the deadline time of the cloud workload using [Eq. 4] and [Eq. 5] in the given budget. Allocate resources based on time, the workload which has shortest deadline time will execute first. If the two workloads have same deadline time then that workload will execute first that has lesser execution time [Eq. 2]. Resource scheduler schedules all the cloud workloads with smallest execution time workload to the resources that provide required QoS. If any deadline is found missed then compensation will be given. In *cost based scheduling policy*, QoS manager calculates the cost of each cloud workload using [Eq. 6] then sort, as the priority is given to the cloud workload which has maximum budget. If two workloads have same budget then that workload will execute first that has lesser execution time. Resource scheduler schedules all the workloads with high budget workload to the resources that provide required QoS. Suppose, cloud consumer selects "cost based scheduling policy". JADE is used to establish the communication among the IaaS components 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. However, following details enable the understanding of the cloud based environment in which the proposed SLA-aware autonomic resource management technique is implemented:

1) Cloud consumer submits their request to user interface (CWMP) that contains the workload description [workload name, workload type, budget, deadline and policy (cost based or time based)] as shown in Figure 9.
2) CWMP is deployed on Aneka Platform (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) Resource scheduler schedules the resources to the workloads based on the resource allocation mechanism as discussed in *Section 3.2.1*.
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) After successful execution of workloads, this further returns the resources to resource pool.
7) At the end, the autonomic unit returns updated experiment data along with processed workload back to the cloud consumer.

We have identified the QoS requirements for every workload, cloud workload type and their QoS requirements have been designed [8] [9] as described in Table 2. Based on QoS requirements, SLA is designed for every workload.

4.1 STAR Execution

STAR has been implemented as a real application and its performance evaluation is presented through a web service i.e. *cloud workload management portal* by considering QoS parameters at service level. Use Case and Sequence diagram is used to describe the execution of STAR as shown in Figure 8 and Figure 9 respectively. After selecting the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

"*workload type*" and "*workload name*", user selects "*desired deadline*" and "*preferred policy*" and enters "*estimated budget*" as shown in Figure 9. Suppose *Performance Testing* workload is selected by user then STAR identifies QoS requirements (SLA Cost and Execution Time) for this workload using Table 2 and selects workload type to find the appropriate resource(s). In *time based scheduling policy*, STAR executes the workloads with minimum execution time and before deadline while in *cost based scheduling policy*, STAR executes the workloads with minimum execution cost. Resource scheduler schedules all the workloads with high budget request to the resources that provide required QoS. Further for SLA, STAR provides four different types of price plans (Forum Plan, Premium Plan, Advance Premium Plan and Customize Configuration) under "price plan management". First three types of price plans (Forum Plan, Premium Plan, and Advance Premium Plan) are fixed price plans based on the current requirements of cloud consumers and in fourth price plan (Customize Configuration), cloud consumer can customize their requirements to execute their workloads by selecting configuration details (Operating System, Memory Usage Duration (Hours/Day and Hourly Rate) and Data Rate). Based on the SLA information, price plan described by cloud consumer and availability of resources, STAR generates a final schedule of execution of workloads automatically and sends scheduling details (start date, end date and estimated budget) back to a particular cloud consumer in the form of "*Reply*". Finally, all other workloads are scheduled on the available resources set based on their policy.

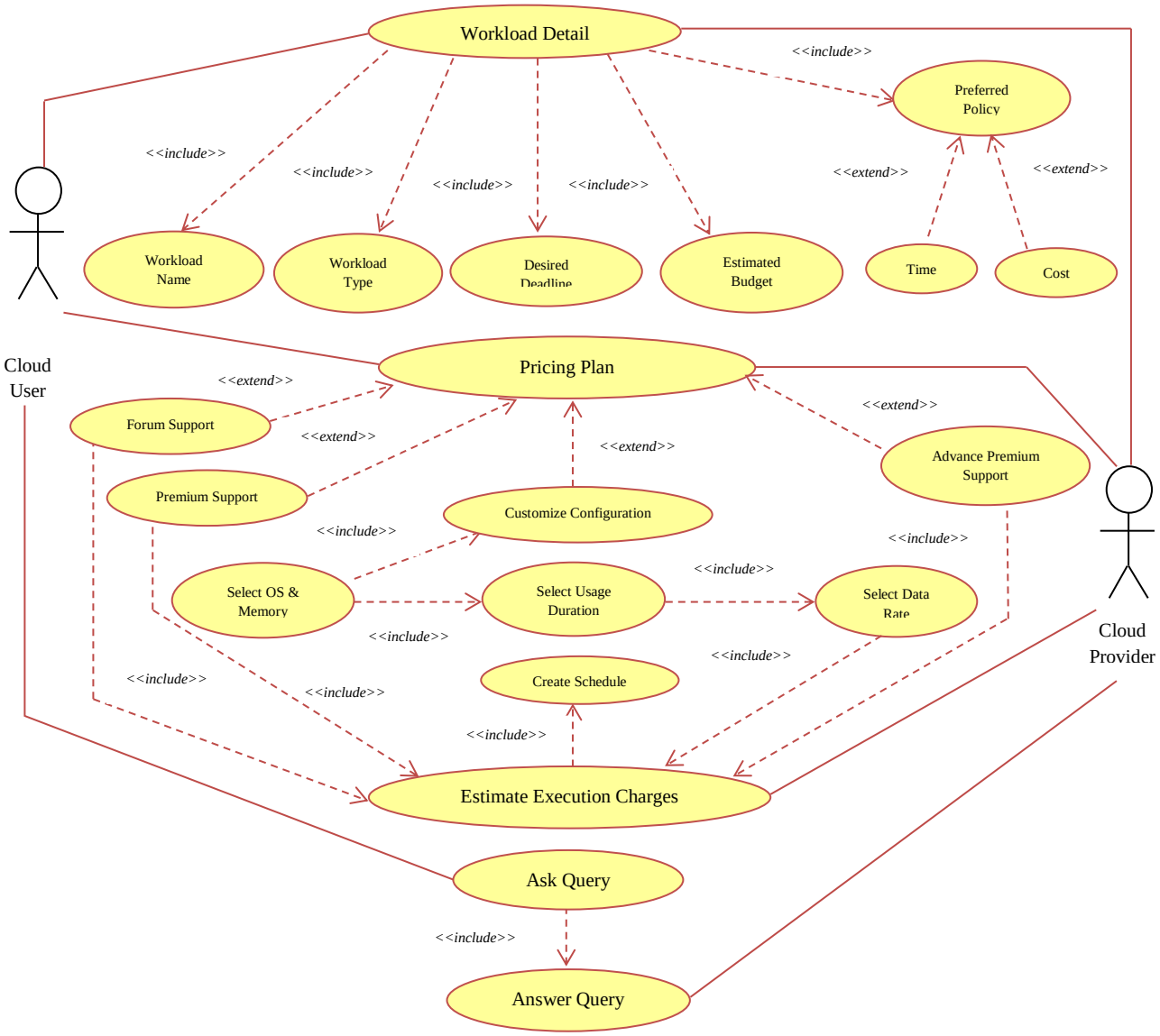

Fig. 8: Use Case Diagram of STAR Execution

4.2 Experimental Results

We have performed different number of experiments by comparing an SLA-aware autonomic resource management technique called STAR with Proactive Resource Allocation (PRA) technique [4] and Partial Utility-driven Resource Scheduling (PURS) technique [5] as described in Section 2. Table 3 describes the range of SVR that is decided to calculate SLA cost by using [Eq. 3] based on the margin (%) by which deadline of workload is missed.

TABLE 3
VARIATION OF SVR WITH DEADLINE MARGIN

SLA Violation Rate (SVR)	Deadline Margin (%)
0.2	1-10.0
0.4	10.1-30.0
0.6	30.1-50.0
0.8	50.1-75.0
1.0	75.1-100.0

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

Test Case 1: Effect of SVR on SLA Cost and Execution Time

We have calculated value of SLA cost for STAR, PRA and PURS with different value of SVR as shown in Figure 10. SLA cost is calculated using [Eq. 3] based on different value of execution time and their deadline margin. STAR performs 7% -12% better than PURS and 8 – 17% better than PRA in terms of SLA cost. At cost = 120 C$, SVR in STAR is 11.9 % lesser than PURS and 15.6% lesser than PRA. Execution time is calculated using [Eq. 2]. As shown in Figure 11, the execution time decreases with decrease in the value of SVR. Value of SVR for STAR, PRA and PURS is calculated. STAR performs 9.6% -14.31% better than PURS and 16.2–21.3% better than PRA.

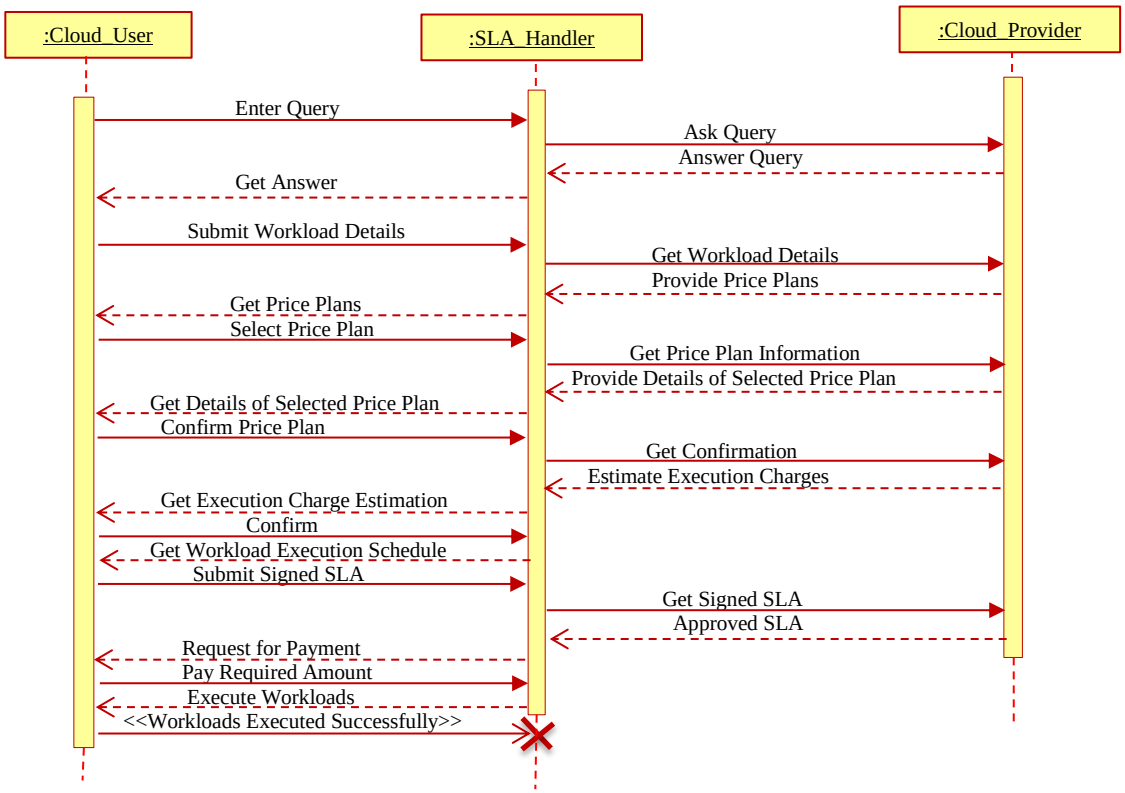

Fig. 9: Sequence Diagram of STAR Execution

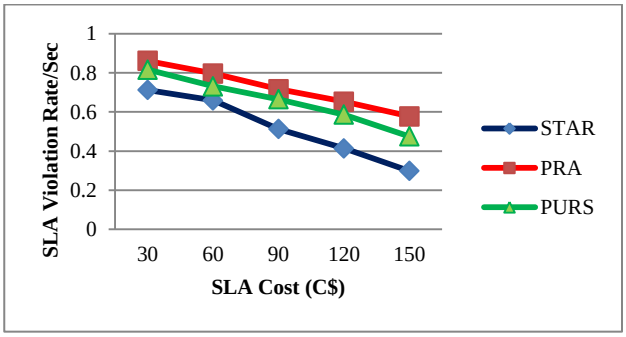

Fig. 10: SLA Violation Rate vs. SLA Cost

All experiments were started with: Workload name: Performance Testing [Processing Larger Image File of Size 713 MB] with two different deadlines and budget (Case 1: Deadline = 700 Seconds, Budget = 55 C$ and Case 2: Deadline = 1800 Seconds, Budget = 250 C$). Table 4 describes the comparison of execution time and execution cost used to process workload on cloud environment for STAR, PRA and PURS. In this experiment, we have considered cloud infrastructures with 8 cores processors to measure the variation of execution time and cost. Table 4 clearly describes that STAR performs better than PRA and PURS with different value of deadline and cost in both the cases.

TABLE 4
SUMMARY OF EXPERIMENT STATISTICS

Scheduling Technique	Budget (C$)	Deadline (Secs)	Execution Time (Secs)	Cost (C$)	SLA Fulfilled
Case 1: Deadline = 700 Seconds, Budget = 55 C$					
PRA	55	700	561	63.36	No
PURS			539	59.11	No
STAR (Cost Based)			509	45.67	Yes
STAR (Time Based)			488	54.91	Yes
Case 2: Deadline = 1800 Seconds, Budget = 250 C$					
PRA	250	1800	1442	227	Yes
PURS			1691	243	Yes
STAR (Cost Based)			1278	203	Yes
STAR (Time Based)			1233	216	Yes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

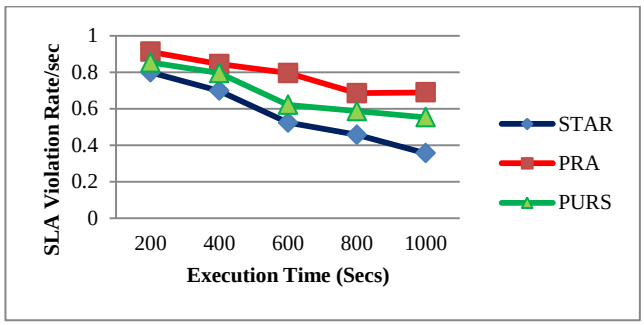

Fig. 11: SLA Violation Rate vs. Execution Time

Test Case 2: Effect of SLA Violation Rate on Number of Workloads and Number of Resources

An experimental result shows that the SLA-aware autonomic resource management technique performs better in terms of SLA Violation rate in two different contexts: i) Number of Workloads (500-3000) and ii) Number of Resources (50-250) in Figure 12 and Figure 13 respectively.

Figure 12 shows STAR performs better than PRA and PURS with increase in number of workloads but performs extraordinary from 1000 workloads to 2500 workloads. It has been depicted from experimental results that SLA violation rate is deceased with increase in number of resources as shown in Figure 13. SLA violation rate is decreasing with increasing number of resources as shown in Figure 13. STAR achieved minimum SLA violation rate at maximum number of resources (250 resources) i.e. SVR = 0.223. STAR performs 4% -9% better than PURS and 7– 13% better than PRA.

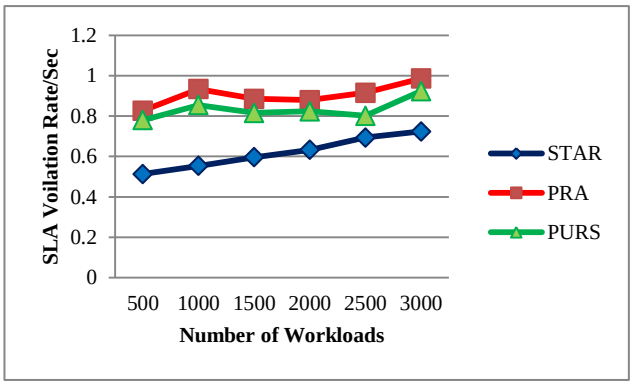

Fig. 12: SLA Violation Rate vs. Number of Workloads

Test Case 3: Effect of SLA Violation Rate on Performance Parameters (QoS)

We have considered six levels of SLA violation rate (0-60%) to test the performance of proposed technique (STAR) by considering five different QoS parameters [2] [8] [9] such as execution time, cost, latency, reliability and availability and compared with PRA and PURS.

Test Case 3.1: Reliability: It is an addition of Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR). We have used following formula [Eq. 7] to calculate reliability.

$$Reliability = MTBF + MTTR \qquad (7)$$

Where Mean Time To Repair (MTTR) is ratio of total downtime to number of breakdowns as shown in [Eq. 8].

$$MTTR = \frac{Total\ Downtime}{Number\ of\ Breakdowns} \qquad (8)$$

Where Mean Time Between Failure (MTBF) is ratio of total uptime to number of breakdowns as shown in [Eq. 9].

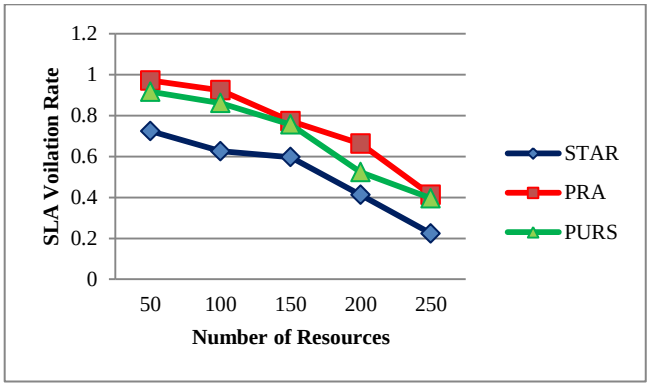

Fig. 13: SLA Violation Rate vs. Number of Resources

$$MTBF = \frac{Total\ Uptime}{Number\ of\ Breakdowns} \qquad (9)$$

Test Case 3.2: Availability: It is a ratio of Mean Time Between Failure (MTBF) to Reliability ($MTBF + MTTR$). We have used following formula [Eq. 10] to calculate availability.

$$Availability = \frac{MTBF}{MTBF + MTTR} \qquad (10)$$

We have calculated percentage of availability for STAR, PRA and PURS. With increasing the SLA Violation Rate, the percentage of availability is decreasing. The percentage of availability in STAR is more as compared to PRA and PURS at different levels of SLA Violation Rate as shown in Figure 14.

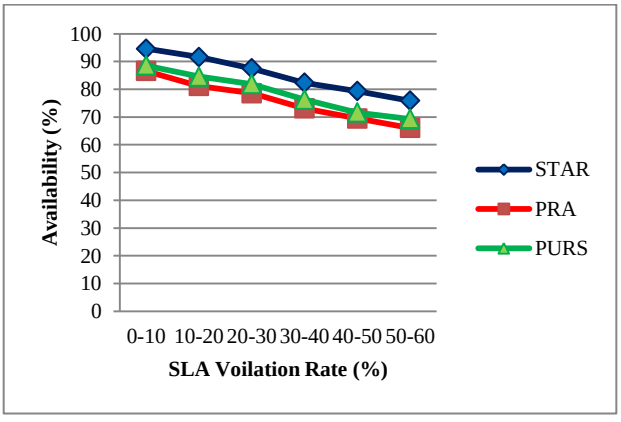

Fig. 14: SLA Violation Rate vs. Availability

The maximum value of availability is 94.62% at minimum SLA Violation Rate. With increasing the SLA Violation Rate, the percentage of reliability is decreasing as shown in Figure 15. The percentage of reliability in STAR is more as compared to PRA and PURS at different levels of SLA Violation Rate. The maximum value of reliability is 27.66% at 0-10% SLA Violation Rate.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

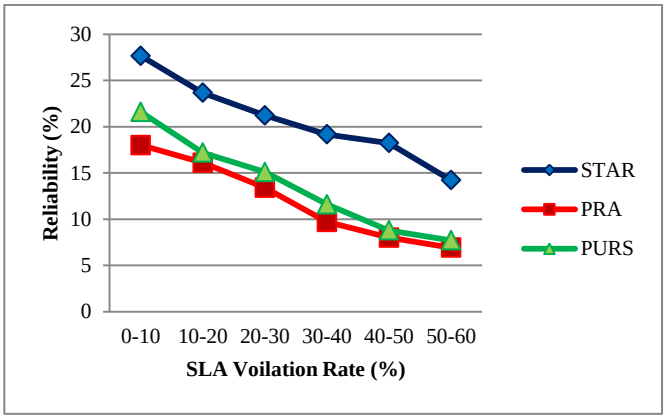

Fig. 15: SLA Violation Rate vs. Reliability

Test Case 3.3: Latency: It is calculated for STAR, PRA and PURS with different levels of SLA Violation Rate. Latency is a defined as difference of time of input cloud workload and time of output produced with respect to that workload. We have used following formula [Eq. 11] to calculate Latency:

$$Latency_i = \sum_{i=1}^{n} (time\ of\ output\ produced\ after\ execution - time\ of\ input\ of\ cloud\ workload) \quad (11)$$

With increasing the value of SLA Violation Rate, the value of latency is increasing. The value of latency STAR is lesser as compared to PRA and PURS at different levels of SLA Violation Rate as shown in Figure 16. The minimum value of latency is 6.14 seconds at 0-10% SLA Violation Rate.

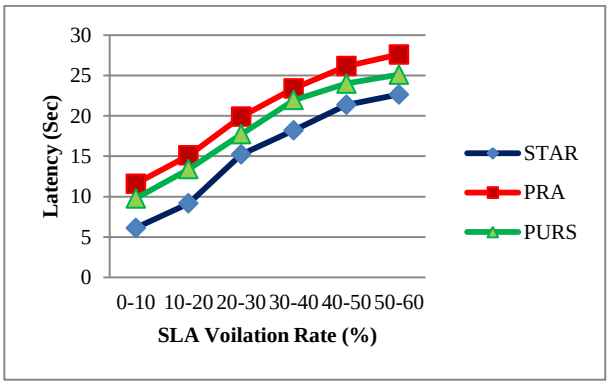

Fig. 16: SLA Violation Rate vs. Latency

Some authors considered latency within the execution time in existing resource management techniques but it is necessary to consider latency separately to test the capability of individual resources using in real time applications.

Test Case 3.4: Execution Time: It is a ratio of difference of workload finish time and workload start time to number of workloads as described in [Eq. 2]. As shown in Figure 17, the execution time is increasing with increase in SLA Violation Rate. At 20-30% SLA Violation Rate, execution time in STAR is 3.72% lesser than PURS and 14.33% lesser than PRA. After 20-30% SLA Violation Rate, execution time increases abruptly but STAR performs better than PRA and PURS.

Test Case 3.5: Execution Cost: It is an addition of resource cost and penalty cost (In case of SLA Violation, different penalty cost for different levels of SLA Violation Rate). STAR defined the different levels of penalty rate based on QoS requirements. Delay time is difference of actual and expected completion time. We have used [Eq. 6] to calculate execution cost. Execution cost is increasing with increase in SLA Violation Rate as shown in Figure 18. At 0 - 10 % SLA Violation Rate, average cost in STAR is slightly lesser than PURS but STAR performs excellent at other levels of SLA Violation Rate as compared to PRA and PURS. An average cost in STAR is 6.69 % lesser than PURS and 11.6% lesser than PRA.

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 framework used for creating the statistics. It states the deviation of the data as a proportion of its average value, and is calculated as follows [Eq. 12]:

$$Coff.\ of\ Var. = \frac{SD}{\mathrm{M}} \times 100 \quad (12)$$

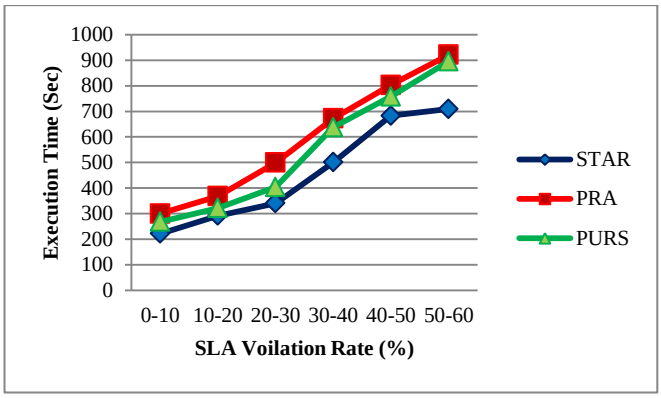

Fig. 17: SLA Violation Rate vs. Execution Time

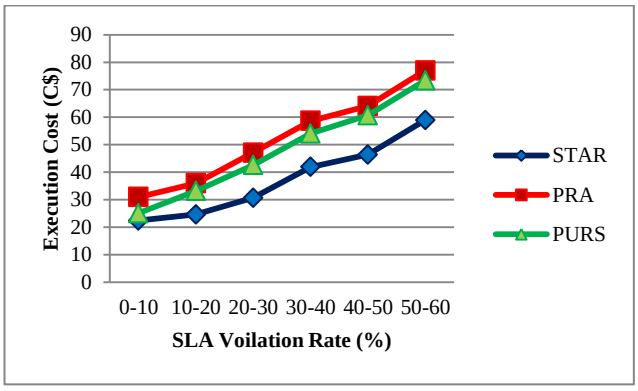

Fig. 18: SLA Violation Rate vs. Execution Cost

Where SD is a Standard Deviation and M is Mean. $Coff.\ of\ Var.$ of SLA violation rate has been studied with respect to number of workloads and number of resources for STAR, PRA and PURS as shown in Figure 19 and Figure 20 respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2017.2648788, IEEE Transactions on Cloud Computing

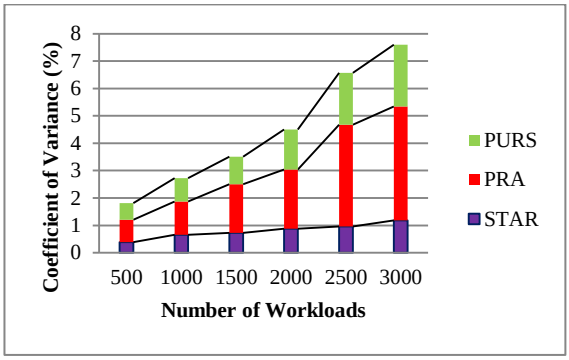

Fig. 19: CoV for SLA Violation Rate with respect to Number of Workloads

Range of $Coff.\ of\ Var.$ (0.39% - 1.18%) for SLA violation rate with respect to number of workloads and (1.36% - 3.91%) for SLA violation rate with respect to number of resources approves the stability of STAR as shown in Figure 19 and Figure 20 respectively. Small value of Coefficient of Variation ($Coff.\ of\ Var.$) signifies STAR is more efficient and stable in resource management in the situations where the number of cloud workloads and resources are changing dynamically. Value of $Coff.\ of\ Var.$ increases as the number of workloads is increasing. STAR attained the best results in the cloud for SLA violation rate has been studied with respect to number of workloads and number of resources.

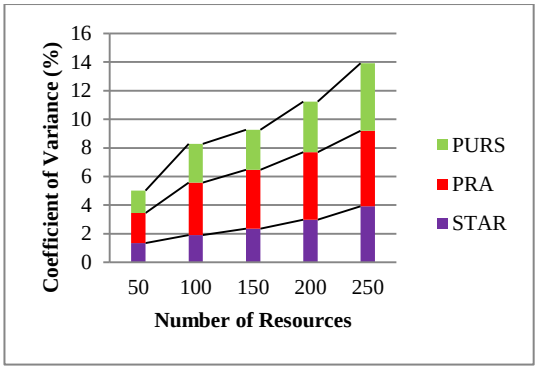

Fig. 20: CoV for SLA Violation Rate with respect to Number of resources

4.4 Discussion

The performance of STAR has been compared with existing resource management techniques (PRA and PURS). Firstly, to test the performance of proposed technique against existing technique, we considered two different cases with different deadlines and budgets to execute same workload. STAR performs 7% -12% better than PURS and 8 – 17% better than PRA in terms of SLA cost. At cost = 120 C$, SVR in STAR is 11.9 % lesser than PURS and 15.6% lesser than PRA. Execution time is calculated using [Eq. 2]. As shown in Figure 12, the execution time decreases with decrease in the value of SVR. Value of SVR for STAR, PRA and PURS is calculated. STAR performs 9.6% -14.31% better than PURS and 16.2– 21.3% better than PRA. Secondly, the performance of STAR has been analyzed with different number of cloud workloads and number of resources, and QoS parameters such as execution time, cost, latency, reliability and availability to analyze the effect of QoS parameters on SLA violation rate. The performance of STAR, PRA and PURS has been evaluated on same cloud environment. We have considered different levels of SLA violation rate (0-60%) to test the performance of STAR. STAR achieved minimum SLA violation rate for maximum number of resources (250 resources) i.e. SVR = 0.223. The maximum value of availability is 94.62% at minimum SLA Violation Rate. The maximum value of reliability is 27.66 at 0-10% SLA Violation Rate. The minimum value of latency is 6.14 seconds at 0-10% SLA Violation Rate. At 20-30% SLA Violation Rate, execution time in STAR is 3.72% lesser than PURS and 14.33% lesser than PRA. After 0-10% SLA Violation Rate, execution time increases abruptly but STAR performs better than SRA. At 20 - 30 % SLA Violation Rate, average cost in STAR is slightly lesser than PURS but STAR performs excellent at other levels of SLA Violation Rate as compared to PRA and PURS. An average cost in STAR is 6.69 % lesser than PURS and 11.6% lesser than PRA. Statistical significance of the results has been analyzed by coefficient of variation, a statistical method for statistical measurement of the distribution of data about the mean value to find the stability of STAR with small value of coefficient of variation. Considering all these QoS parameters and outcomes, it is shown that the STAR delivers a superior autonomic solution for heterogeneous cloud workloads and approximate optimum solution for challenges of autonomic resource management.

5. Conclusions and Future Directions

In this paper, cloud based SLA-aware autonomic resource management technique (STAR) has been proposed for execution of heterogeneous workloads by considering generic property of self-management. The main aim of STAR is to reduce SLA violation rate and improve user satisfaction by fulfilling their QoS requirements. Further, STAR considered different QoS parameters such as execution time, cost, latency, reliability and availability to analyze the impact of QoS parameters on SLA violation rate. The performance of STAR has been evaluated in real cloud environment and the experimental results show that the proposed technique performs better in terms of SLA violation rate as compared to existing resource management techniques.

This paper focused one of the key aspects of self-management, i.e. self-optimization. Future research directions for extending the work to support other characteristics of autonomic systems are:

- STAR can be extended to exhibit properties such as self-healing (find and react to sudden faults), self-configuration (capability to readjust resources) and self-protecting (detection and protection of cyber-attacks).
- STAR 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.
- STAR 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.
- STAR currently considers cost, execution time, SLA violation, availability, reliability and latency QoS parameters. Further, STAR can be enhanced to work with some other parameters also energy efficiency, attack detection rate, resource utilization and resource contention, scalability etc.

Acknowledgements

One of the authors, Sukhpal Singh (SRF-P), gratefully acknowledges the Department of Science and Technology (DST), Government of India, for awarding him the INSPIRE (Innovation in Science Pursuit for Inspired Research) Fellowship (Registration/IVR Number: 201400000761 [DST/INSPIRE/03/2014/000359]) to carry out this research work.

References

[1] Sukhpal Singh and Inderveer Chana, "QoS-aware Autonomic Resource Management in Cloud Computing: A Systematic Review","ACM Computing Surveys" , Volume 48, Issue 3, pp. 1-46, 2015.

[2] Sukhpal Singh, Inderveer Chana, Maninder Singh and Rajkumar Buyya, "SOCCER: Self-Optimization of Energy-efficient Cloud Resources", Cluster Computing [Springer], pp. 1-15, 2016

[3] Buyya, Rajkumar, Saurabh Kumar Garg, and Rodrigo N. Calheiros. "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions." In Cloud and Service Computing (CSC), 2011 International Conference on, pp. 1-10. IEEE, 2011.

[4] Morshedlou, Hossein, and Mohammad Reza Meybodi. "Decreasing impact of SLA violations: A proactive resource allocation approach for cloud computing environments." *IEEE Transactions on Cloud Computing* 2, no. 2 (2014): 156-167.

[5] J. Simao et al, "Partial Utility-driven Scheduling for Flexible SLA and Pricing Arbitration in Clouds", IEEE T. Cloud Computing, 2014

[6] Andrzejak, Artur, Derrick Kondo, and Sangho Yi. "Decision model for cloud computing under sla constraints." In 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 257-266. IEEE, 2010.

[7] Ishakian, Vatche, Raymond Sweha, Azer Bestavros, and Jonathan Appavoo. "Cloudpack* exploiting workload flexibility through rational pricing." In Proceedings of the 13th International Middleware Conference, pp. 374-393. Springer-Verlag New York, Inc., 2012.

[8] Sukhpal, Singh, and Inderveer Chana, 'Q-aware: Quality of Service based Cloud Resource Provisioning", "Computers & Electrical Engineering", [Elsevier], 47, pp. 138–160, 2015

[9] Sukhpal, Singh, and Inderveer Chana. "QRSF: QoS-aware resource scheduling framework in cloud computing." The Journal of Supercomputing, [Springer], Vol. 71, no. 1, pp: 241-292, 2015

[10] García, Andrés García, Ignacio Blanquer Espert, and Vicente Hernández García. "SLA-driven dynamic cloud resource management." Future Generation Computer Systems 31 (2014): 1-11.

[11] Emeakaroha, Vincent C., Marco AS Netto, Rodrigo N. Calheiros, Ivona Brandic, Rajkumar Buyya, and César AF De Rose. "Towards autonomic detection of SLA violations in Cloud infrastructures." Future Generation Computer Systems 28, no. 7 (2012): 1017-1029.

[12] Serrano, Damián, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr, Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens. "SLA guarantees for cloud services." Future Generation Computer Systems 54 (2016): 233-246.

[13] Garg, Saurabh Kumar, Adel Nadjaran Toosi, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya. "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter." Journal of Network and Computer Applications 45 (2014): 108-120.

[14] Casalicchio, Emiliano, and Luca Silvestri. "Mechanisms for SLA provisioning in cloud-based service providers." Computer Networks 57, no. 3 (2013): 795-810.

[15] Zhou, Zhou, Zhigang Hu, and Keqin Li. "Virtual Machine Placement Algorithm for Both Energy-Awareness and SLA Violation Reduction in Cloud Data Centers." Scientific Programming 2016 (2016).

[16] Son, Seokho, and Sung Chan Jun. "Negotiation-based flexible SLA establishment with SLA-driven resource allocation in cloud computing." In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pp. 168-171. IEEE, 2013.

[17] Wu, Linlin, Saurabh Kumar Garg, Rajkumar Buyya, Chao Chen, and Steve Versteeg. "Automated SLA negotiation framework for cloud computing." In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pp. 235-244. IEEE, 2013.

[18] Wu, Linlin, Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. "SLA-based resource provisioning for hosted software-as-a-service applications in cloud computing environments." IEEE Transactions on Services Computing 7, no. 3 (2014): 465-485.

[19] Goudarzi, Hadi, and Massoud Pedram. "Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter." IEEE Transactions on Cloud Computing, (2016).

[20] Son, Seokho, Gihun Jung, and Sung Chan Jun. "An SLA-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider." The Journal of Supercomputing 64, no. 2 (2013): 606-637.

[21] Wu, Linlin, Saurabh Kumar Garg, and Rajkumar Buyya. "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments." In Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on, pp. 195-204. IEEE, 2011.

[22] Rodrigo N. Calheiros, Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds", Future Generation Computer Systems, 28(6), (2012): 861-870, 2012.

[23] Andrieux, Alain, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. "Web services agreement specification (WS-Agreement)." In Open Grid Forum, vol. 128, p. 216. 2007.

Bibliography

Sukhpal Singh joined Computer Science and Engineering Department of Thapar University, Patiala, India, in 2016 as Lecturer. Dr. Singh 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,. Dr. Singh received the Gold Medal in Master of Engineering in Software Engineering. Dr. Singh 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 30 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 five Ph.D thesis have been awarded and four Ph.D. thesis are on-going. She is also working on various research projects funded by Government of India.

Rajkumar Buyya is a Fellow of IEEE, Professor of Computer Science and Software Engineering, Future Fellow of the Australian Research Council, 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 100+, 51000+ 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.