Computers, Materials & Continua
DOI: 10.32604/cmc.2023.032691
Article

Tech Science Press

Formal Modeling of Self-Adaptive Resource Scheduling in Cloud

Atif Ishaq Khan*, Syed Asad Raza Kazmi and Awais Qasim

Department of Computer Science, Government College University, Lahore, 54000, Pakistan
*Corresponding Author: Atif Ishaq Khan. Email: atif.ishaq@gcu.edu.pk
Received: 26 May 2022; Accepted: 27 June 2022

Abstract: A self-adaptive resource provisioning on demand is a critical factor in cloud computing. The selection of accurate amount of resources at run time is not easy due to dynamic nature of requests. Therefore, a self-adaptive strategy of resources is required to deal with dynamic nature of requests based on run time change in workload. In this paper we proposed a Cloud-based Adaptive Resource Scheduling Strategy (CARSS) Framework that formally addresses these issues and is more expressive than traditional approaches. The decision making in CARSS is based on more than one factors. The MAPE-K based framework determines the state of the resources based on their current utilization. Timed-Arc Petri Net (TAPN) is used to model system formally and behaviour is expressed in TCTL, while TAPAAL model checker verifies the underline properties of the system.

Keywords: Formal modeling; multi-agent; self-adaptive; cloud computing

1 Introduction

Cloud computing is a new paradigm [1], showing astonishing growth in the academic and industrial fields. Essential cloud computing capabilities include on-demand self-service, wide area network access, resource pools and measurable services. Cloud computing is bifurcated into deployment and service models. The deployment of the cloud is public: the cloud infrastructure is accessible to the public on the internet and is owned by a cloud provider, private: the cloud infrastructure is managed by a single organization and can exist on-premises or off-premises, hybrid: the cloud infrastructure includes functions for the public and private clouds and communities: cloud infrastructure is used for specific communities. Cloud Computing provides its services via a set of virtual resources (such as servers, virtual machines, etc.) and offer services based on three main service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Based on these service paradigms it not only reduces software development cost deployment effort, generalizes the reusability of all cloud resources. The services offered by the cloud service provider are pay per usage basis. It means the cost of cloud computing depends upon the usage of resources. The assurance of availability of resources with desired satisfaction is required to meet the Service Level Agreement (SLA) between service provider and the client.

In cloud computing, the process of assigning the available resources to user over the internet is referred as resource scheduling [2]. The main considerations in resource scheduling are jobs, virtual

machines, and the physical machine [3]. The resource scheduling is complex [4] due to several reasons, including the dynamic nature of cloud computing, rapid elasticity and multitenant challenges, the correctness of financial and scientific application and complex provisioning, composition, configuration and deployment requirements. Therefore it is hard to plan a design time resource strategy to deal with all dynamic behaviours at runtime. The efficiency and scalability in cloud computing can only be achieved through proper management of resources.

The rapid growth in the demand of cloud computing has produced load on cloud data centers [5] and demands optimal resource allocation and utilization. The main concerns of cloud computing are load balancing and energy consumption [6,7]. Load balancing is the process in which the load is assigned and reassigned among available resources to maximize the throughput of the system. The load balancing is concerned to reduce the cost, energy consumption, and response time to improve resource utilization and performance of overall system [8,9]. The load balancing divides the traffic between all servers, so data can be sent and received without any delay with load balancing [10]. The major goals of load balancing are cost effectiveness, scalability and flexibility and priority. The load balancing algorithms are classified [11] depending on the state of the system and on who initiated the process. Depending on the state of the system the algorithm may be static or dynamic. While based on who initiates the process it depends on the request generated by the sender, receiver, or symmetric. In static load balancing the main focus is the small distributed environment with high internet speed and ignorable communication delay. The dynamic load balancing algorithm is suitable for large distributed environments with less communication delay and execution time. The algorithm that suits cloud computing is dynamic load balancing as the nature of the cloud itself is dynamic [12]. The allocation of resources in a dynamic environment is complex because of the growing number of managed resources as well as an increase in the number of user with different requirements [13]. It is essential to have an automated system capable of adapting to the evolution of changing platforms and requirements to produce desired outcomes. The system needs to scale up or down automatically as per demands. Human intervention can produce solid results, but it will cost time and money, resulting in an inefficient system. A self-adaptive approach, on the other hand, can lead to a situation where capacity is dynamically changed to meet resource demand. A self-adaptive system is an approach based on the MAPE-K [14] to reason and acts accordingly to the quality of service, cloud service information, and delivery information, and make improved resource allocation decisions for every user request. The MAPE-K based agents make rational decisions based on their ability to detect the environmental condition and alter its behaviour or structure. Such an agent has the ability to monitor changes in their existing environment, analyze the state of the system based on the sensed information, plan next possible action based on the state of the system and its own capability and option and executes a plan. While the knowledge of the agent about the environment and itself supports all these steps. Most of the work in cloud computing is simulation-based however the need for formally modeling and verifying an agent-based cloud environment is also very important. Resource Scheduling techniques have been developed for managing the resources in cloud computing. In [15] authors applied the techniques of formal verification to cloud services. They proposed a UPPAAL based method for modeling and verifying Resource Provisioning as Service (RPaaS). Their proposed model focus on the basic requirements of clients i.e., types and available time of resources. The model works for the basic requirements and does not covers the complex attributes. Biographical reactive systems (BRS) were adopted by [16] as a semantic framework for characterising structural and behavioral features of elastic cloud systems. While, at the service and infrastructure levels, they offered elasticity strategies for horizontal scaling and (de)provisioning of cloud resources. The BRS specification in their work is encoded into Maude language to enable automatic executability. Fan et al. [4] proposed an adaptive

resource scheduling strategy for cloud computing. Also the model takes reliability and time into consideration for the modeling process, and Petri nets are used to model the proposed base layer and meta layer models. The adaptive resource scheduling strategy is converted into CTL formulas and the basic properties of the system are analyzed. They performed different experiments to evaluate the various aspects of their proposed method. The hierarchical method based on Petri nets proposed by Fan et al. [17] considers only resource utilization rate and cost. When the number of tasks is increasing, the struggle for resources also increases, and this creates complexity. The prime aim of the scheduling algorithm is to speed up the execution of a task in the cloud. Amiziani et al. [18] discussed the problems of elasticity in cloud computing. They proposed and formalized two operations, Duplication and Consolidation using Petri nets. Souri et al. [19] presented a systematic and comprehensive literature review of the formal verification approaches in cloud computing. The results presented to strengthen the motivation to model and verify the cloud services.

An agent-based self-adaptive system for cloud platform is proposed by [13]. Their model follows the MAPE-K approach to respond according to QoS. Simulations are done to test their model, however formal verification of their proposed model is not done. Although in [16] a biographical reactive system is used to formally model the cloud system. However, the proposed model only considered one aspect which is elasticity. In [20] a dynamic scheduling algorithm is proposed that balances the workload among all virtual machines. However, the workload is balanced based on the deadline and lack any kind of self adaptively. A MAPE-K based framework SMARTS for modeling of self-adaptive real-time multi-agent system is proposed in [21] that defines a set of interfaces for the formal modeling of self-adaptive real-time multi-agent systems. In our previous work [22] an intelligent cloud based load balancing system is proposed. The proposed system makes use of Fuzzy logic to intelligently distribute load among different resources. A self-adaptive load balancing technique proposed in [23] uses live migration for load balancing. In their work a centralized load balancer is installed that monitors the load of every machine and then makes decision of migration. However, their algorithm considers only a single threshold value to decide about migration. An intelligent machine learning and self-adaptive resource allocation framework for cloud computing is proposed in [24]. Their proposed work decreases the energy usage and boost the resource allocation usage time. The run-time decision making for resource allocation is primarily based on improved Bat Algorithm (IBA). Workload-aware efficient resource allocation and load balancing framework for fog computing is proposed in [25]. Their proposed approach is tailored to the internet of health things (IoHT). Task sequencing, dynamic resource allocation, and load balancing are main components of the suggested algorithmic framework. Their approach achieves a 25% reduction in network bandwidth, a 37% reduction in energy usage, and a 45 percent reduction in delay, outperforming existing approaches. However, there are certain limitations to their research, such as security, privacy, and determining the overall computing cost.

It is observed from the literature surveyed to the best of our knowledge that such an agent based self-adaptive system doesn't exist which is characterized by the parameters including, cost, energy, performance, resource provision and SLA all together. We proposed a Cloud-based Adaptive Resource Scheduling Strategy (CARSS) framework that not only considers the QoS aspect of cloud computing but also manages resource scheduling with dynamic load balancing and migrations. The CARSS framework algorithm overcomes the limitation of algorithm presented in [23] by taking not only the threshold but the remaining time to execute and transfer time required as well before deciding the migration and balancing the load. This helps to minimize the migrations and gain better load balancing. The multi-agent approach is used with the MAPE-K to formally model and verify the self adaptivity at all stages of resource scheduling and load balancing. The Int-agent deployed at host

monitors the system locally and make local decisions while share their state for global decision making. Also, to the best of our knowledge a multi agent self-adaptive system is not formally modeled and verified as our proposed CARSS framework is modeled and verified. By using different parameters, the proposed framework is modeled in TAPN and for consistency specifications that CARSS should meet are verified in TAPAAL model checker.

2 Preliminaries

The Self-adaptive resource scheduling is dynamically adapting a strategy without the intervention of humans. In this section we will discuss terminologies required to understand the problem under discussion.

2.1 Self Adaptive System

Self-adaptive systems are able to modify their execution behavior to achieve system goals. The reasons for triggering adaptation action in such system are unpredictable circumstances such as changes in the system‘s environment, system faults, new requirements, and changes in the priority of requirements. In order to deal with these uncertainties, a self-adaptive system continuously monitors itself, gathers data, and analyzes them to decide if adaption is required [26].

2.2 MAPE-K Feedback Loop

The Autonomic Computing proposed by IBM [27] defined four major areas namely Self-Configuration, Self-Healing, Self-Optimization, and Self Protection. The MAPE-K loop in autonomic computing represents Monitor, Analyze, Plan, Execute Phase, and Knowledge base. The Monitor phase is responsible for monitoring the system parameter from the primary stage of execution. The Analyzing phase is responsible for analyzing the adapting symptoms by monitoring the data. The Planning phase is responsible for adaptation strategies while the execution phase is responsible for applying the adaptation strategy. Knowledge is regarded as a central element as it works in integration with all phases to deliver a reliable outcome.

2.3 ARTIS Agent Architecture

An Agent is an entity that perceives an environment through sensors and acts on that environment through effectors [28]. Architecture for Real-Time Intelligent Systems (ARTIS) was originally proposed in [29]. It is an extension of the blackboard model in which many agents cooperate together for solving a general problem. Each ARTIS agent is incorporated with many internal agents (Int-Agent) [30]. An Int-Agent has the necessary knowledge to solve a particular problem and has the additional feature of reactivity, collaborative performance, and hard real-time activity. For these additional features, each Int-Agent consists of reflex level, cognition level, and action level. Further, each Int-Agent consists of reflex level, cognition level, and action level. The nature of an Int-Agent is either critical or non-critical. Each ARTIS agent has a Control Module that is responsible for the realtime execution of each component that belongs to it. The control module is further divided into submodules Reflex Server (RS) which controls the execution of tasks in a critical environment and Deliberate Server (DS) which controls the execution of deliberate tasks.

2.4 Timed-Arc Petrinet (TAPN)

A TAPN is a 7-tuple (P, T, IA, OA, t, i, inv) where P is finite set of places, T is finite set of transitions, IA is a finite set of input arcs, OA is finite set of output arcs, t is a function defining

transport arc, i is a function defining inhibitor arcs, and inv is a function assigning age invariant to places [31].

3 Cloud-Based Adaptive Resource Scheduling Strategy Framework (CARSS)

Self adaptivity allows to adapt according to situational requirement. Our proposed framework Cloud-based Adaptive Resource Scheduling Strategy (CARSS) is shown in Fig. 1. We introduced an agent-based self-adaptive approach that makes use of MAPE-K to schedule the resources cost effectively and energy efficiently. The self-adaptive strategy is based on the analysis of the execution process of user requests from requirement analysis to allocation of the virtual machine and then the accomplishment of the task. Agents are intelligent software entities that have the ability to work autonomously on behalf of user and interact with the environment in context. Self-adaptive agents are based on the MAPE-K feedback loop [14]. These agents can sense environment and then make rational decisions. They also can monitor any change in the environment and take rational decision after analyzing the change and its impact. These agents are designed in such a way that they are loaded with prior knowledge about the environment. The knowledge is updated as the new plans are generated. That‘s the reason, the model works on closed loop manner to constantly perceive the environment and adapt to the changes.

Our proposed framework deploys one ARTIS agent at each data center. An agent in the ARTIS architecture has multiple internal agents (Int-Agent) each of which can solve a specific problem. In proposed framework every host machine is deployed with one Int-Agent. The frame work consists of managed and managing systems. The managed system is responsible for the functional requirements of the system. Each internal agent is composed of MAPE-K based architecture. When a change in environment is observed through sensors, the Int-Agent is notified to complete the task. When a request is completed by these agents a notification is sent back to the environment. Each Int-Agent is connected directly to the knowledge base and updates it regularly. All the updates to the Monitor Int-agent are received through the managed system and all updates are forwarded to managed system by Executer Int-agent once the task assigned is completed. The managed system is responsible to control the hardware attached to it in order to achieve the functional requirement. The Int-Agent on every host manages the virtual machine while the ARTIS agent on data center manages the host. The communication among data centers is controlled through the directory facilitator while communication among hosts is managed by Agent Management System. The deployment of the framework is as per the configuration of the service provider.

It is evident that statistical analysis of historical data to detect an overloaded or under-loaded host is not precise for an environment where the workload is dynamic and unpredictable. In proposed framework the Int-Agent monitors the host and interacts with other Int-Agent deployed on other hosts. All these Int-Agents work cooperatively and update each other about the status of the relevant host. This interaction helps to get updated status of each other and selection of suitable host if migration is required. This kind of interaction and monitoring allows making more accurate decisions to detect overloaded and under-loaded hosts and the selection of appropriate VM for migration. The self-adaptive resource scheduling in CARSS is based on the identification of host overload and underload status. An Int-Agent detects when a host is being overloaded. When a host is overloaded it ultimately suffers a shortage of resources that may lead to SLA violation and QoS degradation. The Int-Agent decided to migrate one or more VM from the overloaded host to another suitable host to overcome the need for resources inside the overloaded host. The Int-Agent also detects when a host is being under-loaded. The detection of the underload host is required to reduce energy consumption.

The Int-Agent decides to migrate all VM from the under-loaded host to another suitable host. Once the VM is migrated the under-load host is switched off to save energy. However, if no suitable host is available for any of the VM the migration will not be done. For migration, the Int-Agent has to select an appropriate VM and determine which host will be suitable for candidate VMs for migration.

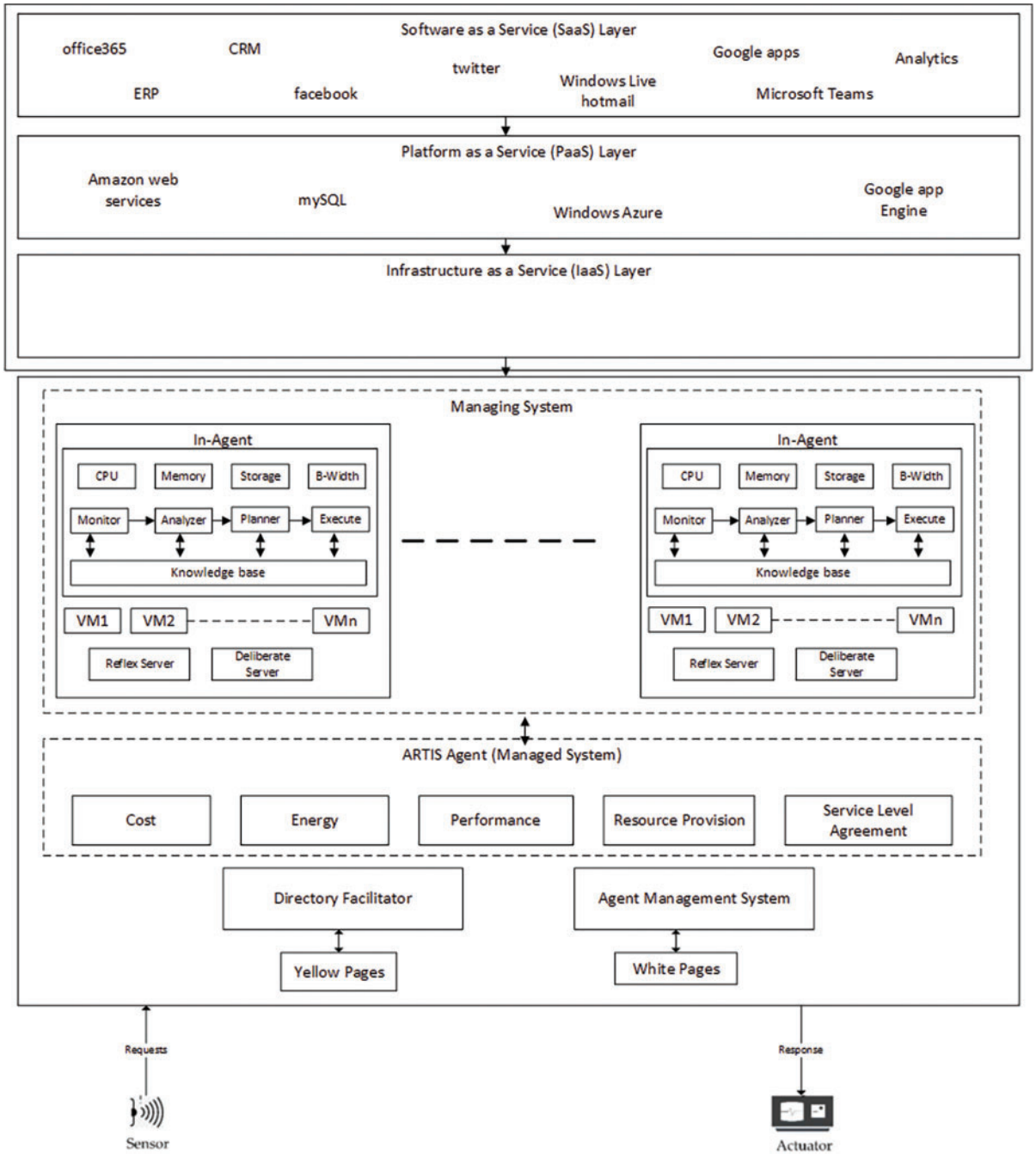

Figure 1: Cloud-based adaptice resource scheduling (CARSS) framework

In CARSS, request is received in SaaS layer. This layer accepts user requests as per SLA and transfer it to PaaS layer. The third layer is IaaS that consists of data center. A data center offers services in the form of virtual machines. The offered resource is either stand alone or composite resource.

A request in our case is $R_q = (R_{id}, type, p, q, m, TL, DL)$ where R_{id} is the unique identifier of a request, *type* is the resource type that is a virtual machine in our case, p is the requested processing speed in Millions Instructions per Second (MIPS), q is the number of processors requested, m is memory requested, TL is the length of a task in Million Instructions (MI), and DL is the deadline of the submitted task.

A resource (R_s) in our case is a virtual machine VM = (id, vs, vl, vr, vq, vt, N) where id is VM identifier, vs is the processing speed of VM in MIPS, vl is the current load on VM, vr is the memory size of VM, vq is the number of processors, vt is the time a resource is available, and N is the total number of requests submitted to a VM.

The proposed self-adaptive strategy of resources in cloud computing is five-tuple consisting of Cost, Energy, Performance, Resource Provisioning, and Service Level Agreement. The CARSS uses cost-aware criteria to measure the effect of resource cost at a certain time unit. The associated time T with each task is the time to complete a single task while the cost in CARSS is the task performed in time T. The proposed framework ensures that a resource is optimally utilized to complete its task and consequently a task will be assigned to a resource having optimality with reference to given environment. Eq. (1) determines the total cost to perform n tasks on one host.

$$C_h := \sum_{i=1}^{n} T_i \tag{1}$$

Whereas total cost to perform all the tasks on all the host at one data center is computed with Eq. (2) where m is the total number of host per data center.

$$C_{dc} = \sum_{j=1}^{m} \sum_{i=1}^{n} T_i^j \tag{2}$$

With the increase in interest and trust of user in cloud computing data centers are expanding rapidly and resultantly the Energy Consumption (E_c) problems become significant. The E_c is related to resource utilization. By increasing the resource utilization in efficient way E_c can be managed. E_c of cloud data center at time t is sum of E_c of all active hosts at data center and is represented by Eq. (3).

$$E_c := \sum_{h=1}^{m} Ph^{(t)} \tag{3}$$

$Ph^{(t)}$is the power of host at time t and is calculated using Eq. (4).

$$Ph^{(t)} = \left(P^{\max} - P^{idle}\right) \times U_{cpu}^{(t)} \tag{4}$$

where P^{idle} and $P^{\max}$ represents the E_c of the host when it is 0% or 100% respectively. $U_{cpu}^{(t)}$ represents CPU utilization of one host at time t that is the main factor affecting E_c. Whereas, $U_{cpu}^{(t)}$ at time t is the ratio of resource usage (RU)to total resources (TR) of host , and given as Eq. (5).

$$U_{cpu}^{(t)} = \frac{RU^{(t)}}{TR^{(t)}} \tag{5}$$

Resource Usage (RU) at one data center is equals to the sum of resource usage of all the hosts deployed at that data center and is represented by $\sum_{j=1}^{m} \sum_{i=1}^{n} RU_{ij}$

Performance (P_r) monitoring at cloud helps to manage cloud environment. P_r at time t is the ratio of total requests completed ($TR_c^{(t)}$) to total requests received ($TR_r^{(t)}$) at any host.

$$P_r = \frac{TR_c^{(t)}}{TR_r^{(t)}} \tag{6}$$

Resource Provisioning is to allocate optimal amount of resources at the right time and consequently adjust resources. Resource provisioning enables to create new instance of resources, enhance the capacity of existing resources or preempt unutilized resources to enhance the system performance. It also makes sure that system will adjust resources in case a resource is overutilized or underutilized. For proper resource provision system load is first need to be computed. Workload of a machine can be computed by dividing the task lengths (T_l) of all tasks of a VM at time t with processing power as in Eq. (7).

$$vl_{vm}^{(t)} = \frac{\sum_{i=1}^{n} T_l}{vs \times vq} \tag{7}$$

Whereas, the total load (T L) of all virtual machines at one host is computed in Eq. (8).

$$TL_h = \sum_{i=1}^{n} vl_{vm}^{i} \tag{8}$$

The required resource for a particular request is assigned on the basis of TL_h. The scaling is made at this stage when provisioning of new resource is demanded. The scaling of resources is either horizontal, meaning more VMs to be added, or vertical, meaning CPU power increases. The resource provision method also computes the CPU‘s degree of relevance that reflects on performance. If task can be executed in parallel manner then resource provision adapts horizontal scaling otherwise vertical.

The SLA guarantees the availability of resources and response time as per agreement. If resource required to execute a task is not available or guaranteed response time is not met then SLA violation is generated. The SLA metric, that indicates the targets for the share of customers as predefined minimum level of service, is determined through Eq. (9). In order to satisfy the QoS of user, the SLA violation should be minimized.

$$SLA = \frac{RR^{(t)} - RU^{(t)}}{RR^{(t)}} \tag{9}$$

The Monitor Int-Agent continuously monitors the system and collects data from various system components and applications in execution. The monitoring process makes a system self-aware. The collected data is passed to the analyzer to understand the system‘s current state, progress, and potential future actions. A set of policies is managed by the Analyzer Int-Agent that can be used to perform adaptation as well as a set of goals that must be completed. One or more actuators carry out specific actions on the system during the execute phase. The combination of policy, target and actuator is our adaptation policy. For example while ensuring the self-adaptive strategy we also want to optimize performance and energy consumption by acting on the frequency of processors. In this scenario, our goal is to maximize performance while conserving energy, and our policy is a process that determines how actuators are activated. Several adaptation policies could be active at the same time, influencing each other‘s decisions. Our self-adaptive framework is managing all these things by cooperation. The performance and energy can be optimized by adapting the policy to use minimum amount of resources needed to accomplish a task. It leads to the possibility of consolidating multiple tasks on the same machine to increase resource utilization and minimize energy consumption by destroying underload machine (after migrating task to other machines) and idle machines.

3.1 Formal Modeling of CARSS Framework

The formal methods approach is an effective way to improve reliability and quality of the system. The proposed framework is modeled using Timed Arc Petri nets that provides strong mathematical and graphical representations to incorporate concurrency, sequential execution, conflicts, determinism, resource sharing, timing information, communication, synchronization, and distribution in the underlying system. The Timed-Arc Petrinet of CARSS framework is represented in Fig. 2, while the TCTL properties are used to verify the correctness of the system. Places are passive entities in Timed Arc Petri Net, whereas transitions are active entities. Processes and attributes make up the CARSS framework's environment.

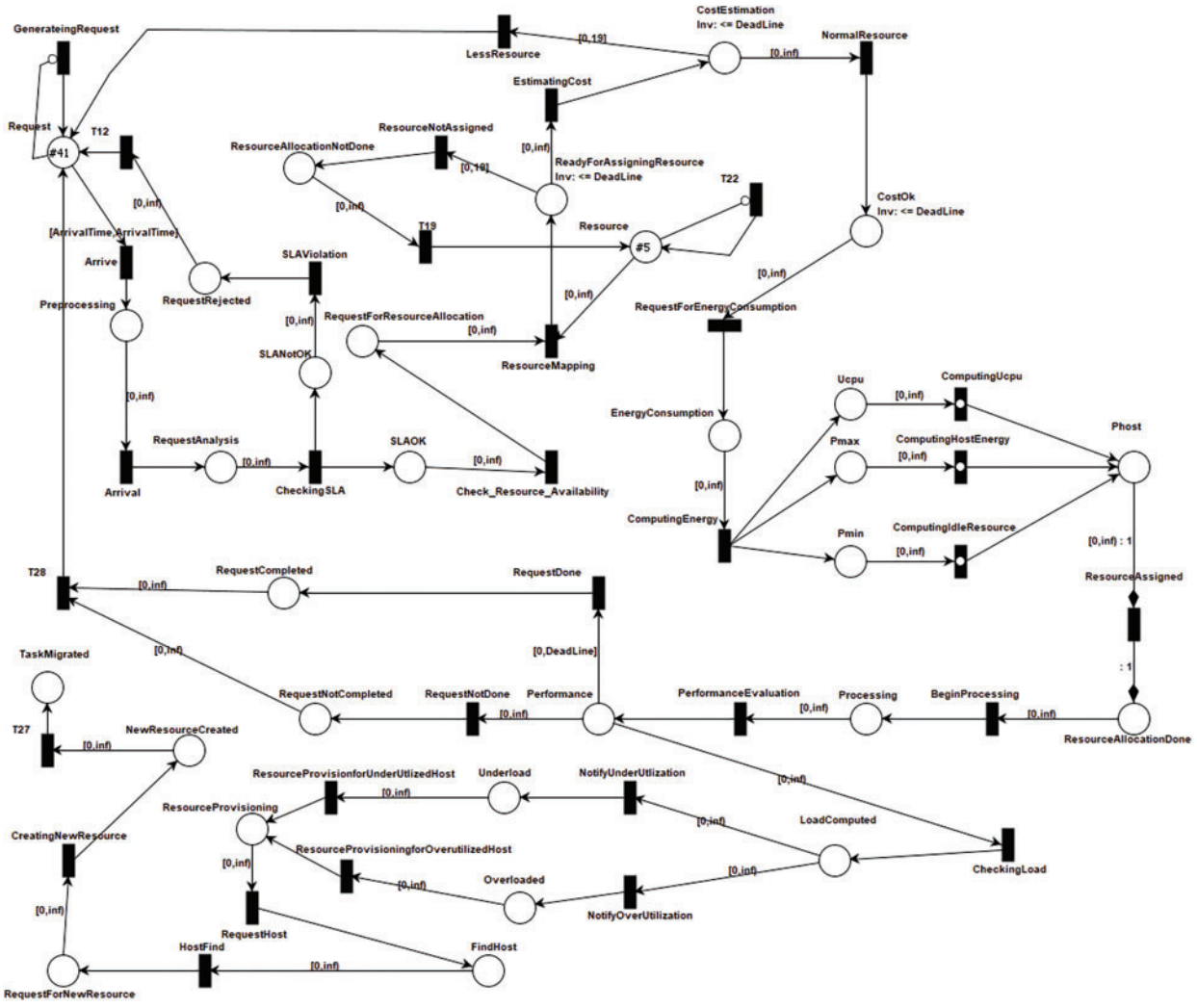

Figure 2: TAPN model of CARSS framework

The attribute values are continuously monitored by agent and when a new event occurs these attribute values are updated. When a request is received, the transition *Arrive* forwards request to *RequestAnalysis* state. The transition *CheckingSLA* forwards request for SLA evaluation. If SLA is not satisfied, the transition *SLAViolation* forwards request to *RequesteRejected* state. While, the request

is forwarded for resource estimation if SLA is not violated. The transition *Check_Resource_Availability* forwards requests to state *RequestForResourceEstimation*, while the transition *ResourceMapping* forwards request to *ReadyForAssigningResource* state. Once the mapping of resource is done the transition *EstimatingCost* forwards request to *CostEstimation* state. The Cost Estimation is to check the total number of task execution in terms of time t. If Cost is satisfied, the transition *ReuestForEnergyConsumption* forwards request to *EnergyConsumption* state. The transition *ComputingEnergy* forwards request to the states U_{cpu}, P_{max}, and P_{min} to compute power of hast at state P_{host}. The power of the host is computed in order to determine energy consumption. The resource is assigned to a particular request if the cost and energy parameter are satisfied. The processing of the request is started, while the transition *PerformanceEvaluation* is enabled to measure the performance of the system. The performance is ratio of number of requests completed to total number of requests received.

When a request is under processing, there are three different possibilities of request completion. The request may be or may not be completed successfully, while the other possibility is that system may change its state to overload or underload. The transition *CheckingLoad* forwards requests to *LoadComputed* state. The transitions *NotifyUnderUtilization* and *NotifyOVerUtilization* forwards request to Underload and Overload states respectively. According the state of the system Resource Provision is done accordingly. The transition *RequestHost* forward a request to find a suitable host. The task is migrated to suitable host in order to balance the overloaded host. While an underload host is switched off after migrating task to suitable host to save energy.

The properties that depicts the behaviour of the system are expressed in TCTL. These properties of the system are verified using TCTL formulas in TAPAAL. The TAPAAL is a verification tool for timed-arc petri-net with in-built verification engine. The proposed framework is verified with a significant number of properties, but for the space limitations, only few verified properties of the system and their corresponding TCTL expressions along with verified results are shown in Tab. 1. The result shows that queries and their corresponding TCTL formulas are satisfied.

Table 1: Properties for correct functioning of CARSS framework and corresponding TCTL formula in TAPAAL

Query	Formula	Result
Is cost and energy computing?	$\exists \Diamond ((\text{CARSS.CostOk} = 1 \wedge \text{CARSS.Energy Consumption} = 1 \wedge \text{CARSS.Ucpu} = 1) \wedge ((\text{CARSS.Pmax} = 1 \vee \text{CARSS.Pmin} = 1) \wedge \text{CARSS.Phost} = 1))$	Satisfied
Is resource allocating to requests?	$\exists \Diamond((\text{CARSS.CostOk} = 1 \wedge \text{CARSS.Energy Consumption} = 1 \wedge \text{CARSS.Ucpu} = 1) \wedge (\text{CARSS.Pmax} = 1 \vee \text{CARSS.Pmin} = 1) \wedge \text{CARSS.Phost} = 1 \wedge \text{CARSS.Resource AllocationDone} = 1)$	Satisfied

(Continued)

Table 1: Continued

Query	Formula	Result
Is request completing?	∃ ◊ (CARSS.SLA = 1 ∧ CARSS.RequestForResourceAllocation = 1 ∧ CARSS.ReadyForAssigningResource = 1 ∧ CARSS.Cost Estimation = 1 ∧ CARSS.CostOk = 1 ∧ CARSS.EnergyConsumption = 1 ∧ CARSS.Pmax = 1 ∧ CARSS.Pmin = 1 ∧ CARSS.Phost = 1 ∧ CARSS.ResourceAllocationDone = 1 ∧ CARSS.Processing = 1 ∧ CARSS.Performance = 1 ∧ CARSS.RequestCompleted = 1)	Satisfied
Is performance evaluating?	∃ ◊(CARSS.CostEstimation = 1 ∧ CARSS.CostOk = 1 ∧ CARSS.EnergyConsumption = 1 ∧ CARSS.Pmax = 1 ∧ CARSS.Pmin = 1 ∧ CARSS.Phost = 1 ∧ CARSS.ResourceAllocationDone = 1 ∧ CARSS.Processing = 1 ∧ CARSS.Performance = 1)	Satisfied
Is resource allocation done and resource over load and not underload?	∃ ◊ (CARSS.ResourceAllocationDone = 1 ∧ CARSS.Processing = 1 ∧ CARSS.Performance = 1 ∧ CARSS.LoadComputed = 1 ∧ CARSS.Overloaded = 1 ∧ CARSS.Underload = 0)	Satisfied
Is host overload and new resource created for migration?	∃ ◊(CARSS.LoadComputed = 1 ∧ CARSS.Overloaded = 1 ∧ CARSS.ResourceProvisioning = 1 ∧ CARSS.FindHost = 1 ∧ CARSS.RequestForNewResource = 1 ∧ CARSS.NewResourceCreated = 1)	Satisfied

3.2 Working of Proposed CARSS Framework

Our proposed framework is more expressive to explain the utilization of resource through MAPE-K. For maximum utilization of resources, reduce energy and better resource provision a novel approach is proposed to determine the state of the resources with MAPE-K control loop for better self-adaptive resource scheduling. The MAPE-K works in closed loop fashion to identify the adaptation needs and defining and executing the operations required to deal with these needs. The sensor is used for communication from the environment to the agent, while the actuator is used for communication from the agent to the environment, as depicted in Fig. 1. The Information is first sensed by the Monitor phase. However, the Executor phase communicates with actuator to respond against request.

The proposed Algorithm 1 demonstrates working of CARSS framework. A new request is sensed from environment based on SLA parameters. The SLA is a user-or resource-specific criterion for each case. New incoming requests are constantly sensed by the Monitor Int-Agent, and a request-id is assigned to each new request. The Monitor Int-Agent forwards request to Analyzer Int-Agent if and only if the request satisfies SLA. The Analyzer Int-Agent evaluates the request for resource mapping and analyze the workload as well to guess utilization. The execution time (ET) for each request is computed to map it to most appropriate R_s^j. It computes the C_h and E_c for each host to find most suitable resource for received requests. The R_s^j is assigned to the R_q^i if execution time of R_q^i at R_s^j ($ET_{R_q^i R_s^j}$) is less than or equals to the required time to execute R_q^i at R_s^j (is computed with $RTE_{R_q^i R_s^j} = DL_{R_q^i} - vl_{R_s^j}$). The Analyzer Int-Agent forwards information to Planner Int-Agent for a suitable adaptation policy. As the nature of the cloud is unpredictable, the Planner phase can add new strategies and update knowledge base accordingly. The Executer Int-Agent takes action based on updated plan in planner phase. The Executer Int-Agent either directly execute an action specified by the Planner Int-Agent or balances the load as per load balancing policy. The workload will be updated at both stages i.e., when task is assigned to a resource and when it completes its execution. As the proposed framework is multi-agent, the internal agents of each agent are internally interconnected to internal agents of the other agents. This interconnectivity enables them to share updated information with each other.

The P_r of the system is constantly monitored by the framework in order to determine the ratio of completed tasks to the received tasks. The workload of each resource ($vl_{R_s^j}^{(t)}$) is constantly monitored and computed at regular time intervals. Since every R_s has upper (UT) and lower (LT) thresholds, while a resource is said to be *underload* when $vl_{R_s^j}^{(t)}$ is less than LT, *overloaded* when $vl_{R_s^j}^{(t)}$ is more than UT, and *balanced* when the $vl_{R_s^j}^{(t)}$ lies between LT and UT. Once a resource is triggered as *overload*, the system finds a suitable *underload* resource where the R_q^i may be migrated. For R_q^i executing on an *overload* R_s^j, system computes remaining execution time of R_q^i at R_s^j ($RET_{R_q^i R_s^j}$) and required transfer time ($TT_{R_q^i R_s^k}$) to migrate R_q^i to R_s^k. The R_q^i is assigned to R_s^k if and only if $RET_{R_q^i R_s^j}$ is less than $TT_{R_q^i R_s^k}$. If $RET_{R_q^i R_s^j}$ is more than $TT_{R_q^i R_s^k}$ then the system may keep the task executing on the same resource. It will save the migration cost as migrating the task to other resource will increase the overall completion time for R_q^i. On the other hand, for energy saving purpose, it is possible that system selects and *underload* resource and migrate it to another host satisfying the same condition as in case of *overload* resource. The *underload* host switched off after migration. In any case, for an overloaded resource there is no suitable resource available for migration the system requests for the creation of new resource enabling elasticity in the system. The tasks are migrated to the newly created host in order to balance the load of the system.

Algorithm 1: CARSS Algorithm

1: **Input :** $R_q\left(R_q^1,\ R_q^2,\ \ldots\ldots,\ R_q^n\right)$
$R_s\left(R_s^1,\ R_s^2,\ \ldots\ldots..,\ R_s^m\right)$
2: **While** true **do**
3: $\forall R_q^i \in \left(R_q^1, R_q^2,\ \ldots..,\ R_q^n\right)$
4: **IF** $(R_q^i \vDash SLA)$ **THEN**
5: $\forall\ R_s^j \in \left(R_s^1, R_s^2,\ \ldots\ \ldots,\ R_s^m\right)$ //R_s is a VM in this case
6: Compute $ET_{R_q^i R_s^j}$ $//ET_{R_q^i R_s^j} = \left(\frac{TL_{R_q^i}}{vs\ \times\ vq}\right)$

(Continued)

Algorithm 1: Continued

Compute C_h for R_q^i
Calculate $U_{cpu}^{(t)}$ to compute $Ph^{(t)}$
Compute E_c for R_q^i at R_s^j
END IF
$R_S^j \leftarrow R_q^i \Leftrightarrow C_h \& E_c$ satisfies & $ET_{R_q^i}$ at $R_s^j \leq RTE_{R_q^i R_S^j}$
Update $vl_{R_s^j}$ $//vl_{R_s^j} = vl_{R_s^j} + ET_{R_q^i}$
Monitor the P_r to determine $\frac{TR_c^{(t)}}{TT_R^{(t)}}$
$\forall R_s^j \in \left(R_s^1, R_s^2, \ldots \ldots, R_s^m\right)$
$\forall R_q^i \in \left(R_q^1, R_q^2, \ldots .., R_q^n\right)$
Calculate $vl_{R_s^j}^{(t)}$
IF $vl_{R_s^j}^{(t)} \geq UT$ **Then** $R_s^j \leftarrow Overload$
ELSEIF $vl_{R_s^j}^{(t)} \leq LT$ **Then** $R_s^j \leftarrow Underload$
ELSE $R_s^j \leftarrow balanced$
$\forall R_s^j \in Overload$ Select R_q^i , compute $RET_{R_q^i R_s^j}$
$\forall R_s^k \in Under - load$, compute $TT_{R_q^i R_s^k}$
$\binom{R_q^i}{R_s^k}$ for migration $\Leftrightarrow RET_{R_q^i R_s^j} < TT_{R_q^i R_s^k}$
Otherwise let R_q^i execute on R_s^j
$\forall R_s^j \in Under - load$, Select R_q^i, compute $RET_{R_q^i R_s^j}$
$\binom{R_q^i}{R_s^k}$ *for migration* $\Leftrightarrow$ $RET_{R_q^i R_s^j} < TT_{R_q^i R_s^k}$ & $R_s^k \in normal$
Migrate the R_q^i to R_s^k and switched off R_s^j for energy saving
$\exists R_q^i$ executing on overloaded host and no suitable host found for migration
Create a new R_s and migrate R_q^i

The proposed CARSS framework selects a R_s with highest acceptance probability and then updates CPU utilization and workload of that particular resource. Secondly based on the shared information, the framework selects best option of migrating a virtual machine to other physical machine. It is important to mention that simultaneous migration of several VMs to reduce energy consumption can cause massive overhead and service degradation. To address these issues our proposed framework collects information from physical machines while allocation and migration is done self adaptively by considering the relevant concerns. When a migration is done the selected VM will empty the physical machine so the physical machine may put in sleep mode for energy saving and migration keeps the CPU utilization of Physical machine below the threshold value. Our self-adaptive system will decide when to create new VM and when to kill VM with respect to allocation and migration in order to enhance the performance of the system and minimize the energy consumption.

4 Conclusion

In this paper the CARSS framework is proposed for self-adaptive resource scheduling for dynamic cloud environment and unpredictable workload which depicts the improved performance. Our paper applies formal modeling and verification to cloud environment in order to determine the correctness

of the system. The proposed CARSS framework is more expressive than the traditional approaches, since it mainly includes managed and managing system. The managed system is an ARTIS agent that works on five parameters namely cost, energy, performance, resource provision and SLA. On the other hand, the managing system deploys Int-Agent at each host for self-adaptive resource scheduling. These In-Agents are internally interconnected. The Int-Agent monitors the system locally and make local decision at their own, while the information is shared among all Int-Agents that plays a vital role for making global decisions like migration and load balancing. In the planner phase determines the state of the system based on current CPU utilization and workload and acceptable adaptation policies are generated for that particular state. The Executer phase implements best strategy received from the Planner phase. The accuracy and preciseness of the functionality of the system is modeled in TAPN and are verified through TAPAAL model checker. The results of the verification shows that our proposed CARSS framework satisfies all properties, demonstrating the rationality and reliability of this framework.

Acknowledgement: Thanks to our families & colleagues who supported us morally.

Funding Statement: The Authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, pp. 50, 2009.

[2] H. Zhao, M. Pan, X. Liu, X. Li and Y. Fang, "Optimal resource rental planning for elastic applications in cloud market,"in *Proc. 2012 IEEE 26th Int. Parallel and Distributed Processing Symp.*, Shanghai, China, pp. 808–819, 2012.

[3] L. Liu, M. Zhang, Y. Lin and L. Qin, "A survey on workflow management and scheduling in cloud computing," in *Proc. 14th IEEE/ACM Int. Symp. on Cluster, Cloud, and Grid Computing(CCGRID)*, Chicago, IL, USA, pp. 837–846, 2014.

[4] G. Fan, H. Yu and L. Chen, "A formal aspect-oriented method for modeling and analyzing adaptive resource scheduling in cloud computing," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 281–294, 2016.

[5] M. AlaŠAnzy and M. Othman, "Load balancing and server consolidation in cloud computing environments: A meta-study," *IEEE Access*, vol. 7, pp. 141868–141887, 2019.

[6] Y. Jadeja and K. Modi, "Cloud computing concepts, architecture and challenges," in *Proc. Int. Conf. on Computing, Electronics and Electrical Technologies [ICCET]*, Tamil Nadu, India, pp. 877–880, 2012.

[7] C. Preist and P. Shabajee, "Energy use in the media cloud: Behaviour change, or technofix?," in *Proc. IEEE Second Int. Conf. on Cloud Computing Technologies and Science*, Indianapolis, Indiana USA, pp. 581–586, 2010.

[8] P. Singh, P. Baaga and S. Gupta, "Assorted load balancing algorithms in cloud computing: A survey," *International Journal of Computer Applications*, vol. 143, no. 7, pp. 34–40, 2016.

[9] S. Goyal and M. K. Verma, "Load balancing techniques in cloud computing environment: A review," *International Journal of Advance Research in Computer Science and Software Engineering*, vol. 6, no. 4, pp. 583–588, 2016.

[10] I. Ivanisenko and T. Radivilova, "Survey of major load balancing algorithms in distributed system," in *Proc. 2015 Information Technologies in Innovation Business Conf. (ITIB)*, Kharkiv, Ukarain, pp. 89–92, 2015.

[11] D. Kashyap and J. Viradiya, "A survey of various load balancing algorithms in cloud computing," *Scientific & Technological Research*, vol. 3, no. 11, pp. 115–119, 2014.
[12] A. More, K. Anurag, R. Kolhe, K. Rupesh and Y. Prashant, "Sla driven load balancing for web applications in cloud computing environment," *Information and Knowledge Management*, vol. 1, no. 1, pp. 5–13, 2011.
[13] M. Soltane, Y. Cardinale, R. Angarita, P. Rosse, M. Rukoz *et al.,* "A Self-adaptive agent-based system for cloud platforms," in *Proc. 3rd Int. Conf. on Pattern Analysis and Intelligent Systems (PAIS)*, Tebessa, Algeria, pp. 1–8, 2018.
[14] D. G. D. L. Iglesia and D. Weyns, "Mape-k formal templates to rigorously design behaviors for self-adaptive systems," *ACM Transaction on Autonomous and Adaptive System*, vol. 10, no. 3, pp. 1–31, 2015.
[15] W. Zhou, L. Liu, S. Lü and P. Zhang, "Toward formal modeling and verification of resource provisioning as a service in cloud," *IEEE Access*, vol. 7, pp. 26721– 26730, 2019.
[16] K. Khebbeb, N. Hameurlain, F. Belala and H. Sahli, "Formal modelling and verifying elasticity strategies in cloud systems," *IET Software*, vol. 13, no. 1, pp. 25–35, 2019.
[17] G. Fan, L. Chen, H. Yu and D. Liu, "Formally modeling and analyzing cost-aware job scheduling for cloud data center," *Software: Practice and Experience*, vol. 48, no. 9, pp. 1536–1559, 2018.
[18] M. Amziani, T. Melliti and S. Tata, "Formal modeling and evaluation of stateful service-based business process elasticity in the cloud," in *Proc. on the Move to Meaningful Internet Systems: OTM 2013*, Berlin, Heidelberg, pp. 21–38, 2013.
[19] A. Souri, N. J. Navimipour and A. M. Rahmani, "Formal verification approaches and standards in the cloud computing," *Computer Standards & Interfaces*, vol. 58, pp. 1–22, 2018.
[20] M. Kumar and S. Sharma, "Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment," *Computers & Electrical Engineering*, vol. 69, pp. 395–411, 2018.
[21] A. Qasim and S. A. R. Kazmi, "Mape-k interfaces for formal modeling of real-time self-adaptive multi-agent systems," *IEEE Access*, vol. 4, pp. 4946–4958, 2016.
[22] A. I. Khan, S. A. R. Kazmi, A. Atta, M. F. Mushtaq, M. Idrees *et al.,* "Intelligent cloud based load balancing system empowered with fuzzy logic," *Computers, Materials & Continua*, vol. 67, pp. 519–528, 2021.
[23] V. Lavanya, M. Saravanan and E. P. Sudhakar, "Self-adaptive load balancing using live migration of virtual machines in cloud environment," *Webology*, vol. 17, no. 2, pp. 735–745, 2020.
[24] M. S. Hasan, E., Balamurugan. S. A. Almamun and K. Sangeetha, "An intelligent machine learning and self adaptive resource allocation framework for cloud computing environment," *EAI Endorsed Transactions on Cloud Systems*, vol. 6, no. 18:e2, pp. 1–14, 2020.
[25] S. Khan, I. A. Shah, N. Tairan, H. Shah and M. F. Nadeem, "Optimal resource allocation in fog computing for healthcare applications," *Computers, Materials & Continua*, vol. 71, no. 3, pp. 6147–6163, 2022.
[26] S. Mahdavi-Hezavehi, P. Avgeriou and D. Weyns, "A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements," in *Managing Trade-Offs in Adaptable Software Architectures*, Boston: Morgan Kaufmann, pp. 45–77, 2017.
[27] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[28] S. Russell and P. Norvig, "Intelligent agents," in *Artificial Intelligence: A Modern Approach*, 3rd ed., USA: Prentice Hall, pp. 34–59, 2010.
[29] A. García-Fornes, A. Terrasa, V. J. Botti and A. Crespo, "Analyzing the schedulability of hard real-time artificial intelligence systems?," *Engineering Applications of Artificial Intelligence*, vol. 10, pp. 369–377, 1997.
[30] V. J. Botti, C. Carrascosa, V. Julián and J. Soler, "The artis agent architecture : Modelling agents in hard real-time environments," in *Proc. European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, Berlin, Heidelberg, pp. 66–73, 1999.
[31] L. Jacobsen, M. Jacobsen, M. H. Moller and J. Srba, "Verification of timed-arc petri nets," in *Proc. Theory and Practice of Computer Science. SOFSEM 2011*, Berlin, Heidelberg, pp. 46–72, 2011.