Applied Intelligence (2019) 49:3464–3489
https://doi.org/10.1007/s10489-019-01454-z

Design and analysis of distributed load management: Mobile agent based probabilistic model and fuzzy integrated model

Moazam Ali[1] · Susmit Bagchi[1]

Published online: 13 April 2019

Abstract
In large-scale distributed systems, performing load monitoring and load balancing is a challenging task in terms of load management. In order to enhance the overall system performance, we have developed and implemented two different models for large-scale distributed load management. The mobile agent-based system is based on a probabilistic normed estimation model. This model uses mobile agents for collecting the instantaneous status of currently available node resources autonomously. The mobile agent is goal oriented and consumes less network and system resources, which is ideal for load monitoring for large-scale distributed systems. Moreover, we have proposed an integrated load balancing and monitoring model for distributed computing systems employing type-1 fuzzy logic. Furthermore, we have proposed a smooth and composite fuzzy membership function in order to model fine-grained load information in a system. In this paper, a detailed software architectural design for mobile agent based load monitoring system as well as the fuzzy-based load balancing approach are presented. The experimental evaluation is presented to compare the behavior and performance of the mobile agent-based probabilistic model and fuzzy integrated model under different load conditions. A detail comparative analysis is presented for the mobile agent-based probabilistic model and fuzzy integrated model to show the performance and efficiency of each model. In this paper, we have computed cross-correlation to find the relation between our proposed models (FIM and MABMS).

Keywords Distributed systems · Mobile agents · Load monitoring · Resource utilization · Cloud computing

1 Introduction

Load management is challenging in large-scale distributed systems such as grid, cloud and enterprise computing. The reason is these computing environments are heterogeneous and scattered geographically [1]. In large-scale distributed systems improving the performance and better utilization of shared resources depend upon the proper load balancing and load monitoring mechanisms [2, 3]. The load monitoring mechanism plays a very important role in identifying the currently available resource status of a node to avoid underloaded or overloaded conditions [4]. Without load monitoring, it is difficult to employ load balancing approaches to distributed systems. It is highly challenging for a system administrator to efficiently monitor system load due to dynamic load management in highly scalable computing environments [5]. Moreover, load balancing algorithms are used to balance the overall workload from over-loaded nodes to under-loaded nodes to ensure system-wide optimum performance. Therefore, a mechanism is required to overcome these problems for determining the dynamics of the resource availability of nodes in distributed systems. In distributed computing, various approaches have been proposed for load balancing. The application of fuzzy logic in designing load balancing algorithms is an efficient load balancing approach [6, 7]. Fuzzy logic is composed of fuzzy sets and rules depending upon non-crisp fuzzy sets to model and make decisions based on uncertainties [8]. Furthermore, mobile agents are used to autonomously manage, design, implement and maintain distributed systems [9, 10]. Mobile agents are autonomous software entities having the ability to migrate through the network from node to node [11–13]. The basic characteristics of mobile agents are autonomy in behavior, social interaction, reactive to its environmental changes and goal driven execution [14]. A mobile agent created in one node can transport its "code" and "state" to another node in the network, where it continues its execution [15–17]. The function of "code" is to start execution and the "state" determines the actions of a mobile agent

✉ Susmit Bagchi
profsbagchi@gmail.com

[1] Department of Aerospace and Software Engineering (Informatics), Gyeongsang National University, Jinju 660701, South Korea

in the destination node. Thus, integrating fuzzy logic with a mobile agent is an attractive approach for load monitoring and load balancing in large-scale distributed systems.

1.1 Motivation

In order to achieve better performance and proper utilization of the currently available resources load balancing and load monitoring approaches are employed in distributed systems [18]. The researcher have taken into account a number of approaches for load monitoring and load balancing in distributed systems [6–8, 11, 19]. Load balancing and monitoring approaches are used to minimize the processing workload on a particular node will result in enhancing the overall system performance and efficiency. To monitor the shared resources and balance the processing workload of large-scale distributed systems such as grid, cluster and cloud have focused on better utilization of resources [20–22]. However, in large-scale distributed systems load monitoring and load balancing approaches have received less attention in terms of decreasing intercommunication, time complexity, autonomous and accuracy of decision-making. On the contrary, we argue that the mobile agent-based load monitoring approaches have several advantages such as [22, 23], (a) Reduced network load, (b) Network delay resolve, (c) Dynamic adaptation, (d) Fault tolerance and, (e) Goal-driven behavior. We have also presented a hybrid architecture for load monitoring and balancing employing fuzzy logic as well as mobile agents. The advantages of our proposed hybrid architecture are intelligent monitoring, accurate decision making, low response time and high throughput.

In this paper, we present the design, implementation, and evaluation of mobile agents based load monitoring as well as a fuzzy integrated model in a distributed system environment. The process load estimation is computed by employing a joint probability model and norm function, which is computationally inexpensive. Moreover, the decision-making process is computed by the fuzzy logic decision module to increase the efficiency and performance of the overall system.

The contributions made in this paper are as follows.

- Designing an autonomous load monitoring model using mobile agents based on the probabilistic norm.
- Updating real-time load information to monitoring node for decision making based on time intervals.
- Adaptive decision making by mobile agents depending on the varying status of a node.
- Type-1 fuzzy logic and mobile agent based integrated model design.
- Comparative analysis between the mobile agent model and integrated agent-fuzzy hybrid model.

Rest of the paper is organized as follows. Section 2 presents related work. Section 3 illustrates designing monitoring agent. Section 4 presents designing fuzzy load balancer. Section 5 presents monitoring and decision algorithms. Section 6 describes the implementation environment. Section 7 represents experimental evaluations. Section 8 presents a comparative analysis of our proposed algorithms with each other and, with other contemporary designs. Lastly, Section 9 concludes the paper.

2 Related work

There are various approaches available for load monitoring and load balancing in distributed systems. In general, to perform load monitoring and load balancing in large-scale distributed systems require a considerable amount of computational resources [2]. The taxonomy of various agent-based monitoring models is illustrated in Fig. 1. Iosup, et al. proposed a monitoring architecture for control of grids, which is known as Toytle [24]. The core issues addressed by Toytle are grid-awareness, scalability and communication standards. Toytle architecture inherits Global Grid Forum Grid Monitoring Architecture (GGFGMA) guidelines [24]. This architecture consists of three layers such as a Distributed core layer, a Hierarchical connection layer, and Local monitor layer. This mechanism ensures that the hierarchical connection yields a highly scalable approach, where the data can be collected from different and large-scale distributed systems with some degree of fault tolerance. Pivot Tracing is a monitoring framework for distributed systems that addresses two important limitations [25]. The first limitation is that most of the monitoring system information is recorded a priori. The second limitation is that the information is stored in a component or machine centric way which makes it very difficult to correlate events that cross these boundaries. In Pivot tracing approach dynamic instrumentation combine with relational operator gives users run-time ability to define arbitrary metrics and to select, filter and group-based meaningful events for other parts in the system. Dan Gunter et al. have proposed a very lightweight instrumentation system for dynamic monitoring of high performance distributed applications (DMHPDA) [26]. The system is used to collect and aggregate detailed monitoring information from distributed applications. The system consists of four main monitoring components such as application instrumentation, activation service, monitoring event receiver, and archive feeder. Moreover, to achieve high-performance none of the above components can cause the pipeline to block while processing the data. InteMon is designed for monitoring and data mining in large clusters [27]. InteMon can monitor more than 100 nodes of a data center. This approach uses the Simple Network Management Protocol (SNMP) and, MySQL database is used for storing monitored data. The advantage of this approach is the ability to automatically analyze the monitored data in real time and to alert users for any anomalies [27]. In recent years, mobile

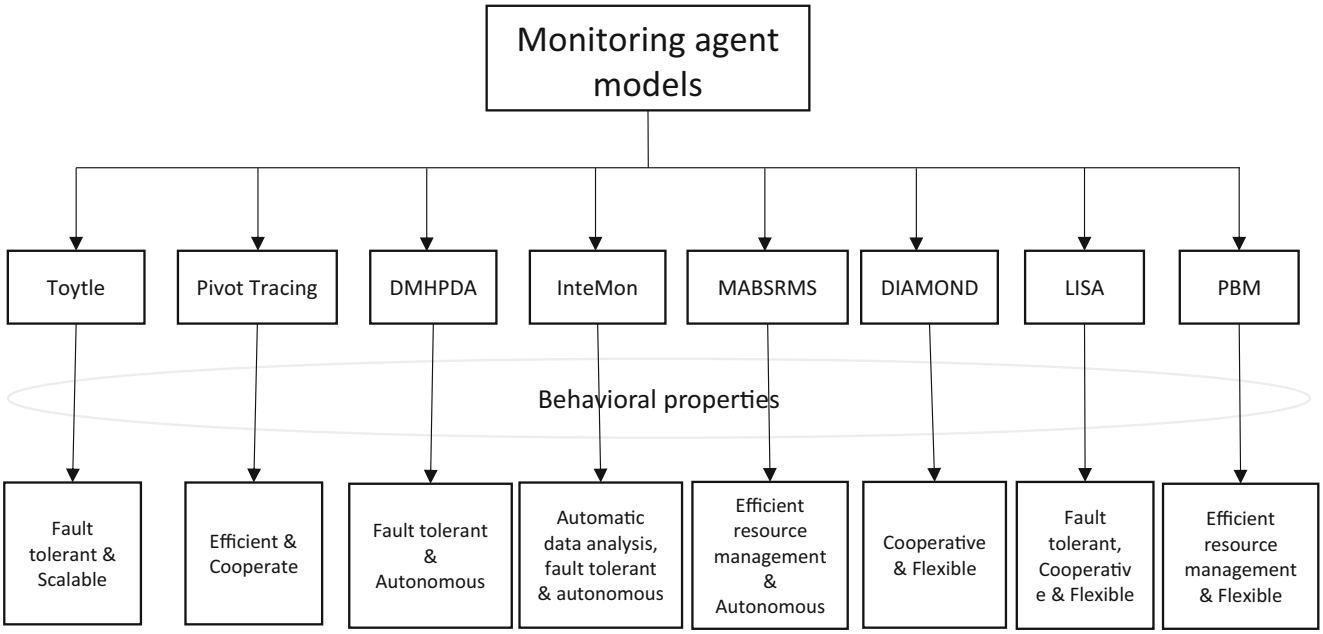

Fig. 1 Taxonomy of various agent-based monitoring systems

agents are emerged as a promising technology for load monitoring and load balancing in large-scale distributed systems. Mobile agents have the ability to migrate from node to node in a connected network to obtain updated status information. Mobile agents are autonomous which means that a user is no longer needed to allocate mobile agents to nodes for migration. The researchers have proposed a Mobile Agent-Based Server Resource Monitoring System (MABSRMS) based on Mobile-C library [28]. In MABSRMS, mobile agents call a low-level function to monitor server resources effectively and efficiently. Monitoring algorithms are easily and quickly deployable on part or all of the monitoring nodes. In MABSRMS there is no backup mechanism specified for the servers. The intercommunication between the nodes and the server would enhance when a large number of nodes are interacting with a single server for load monitoring. Distributed Architecture for Monitoring and Diagnosis (DIAMOND) is a hierarchical and distributed cooperating agent model designed for distributed monitoring and diagnosis system [3]. The hierarchical model gives the advantage to

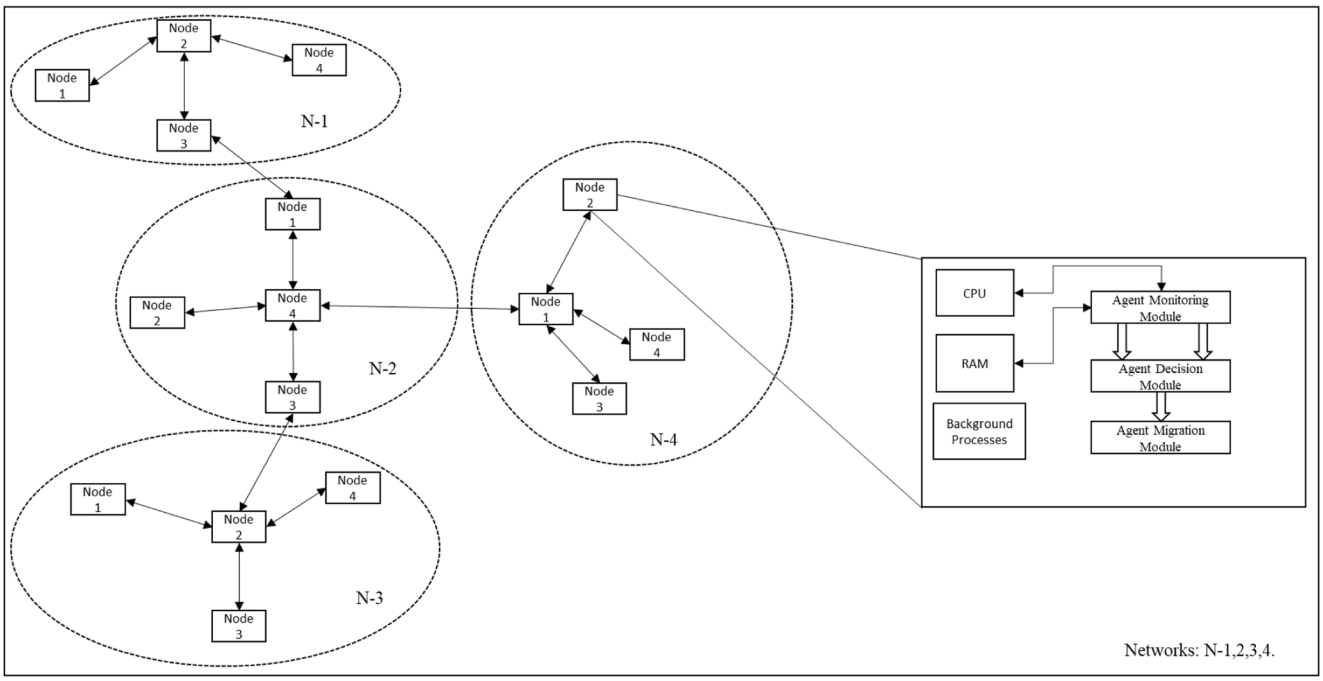

Fig. 2 Large scale distributed system and mobile agent model

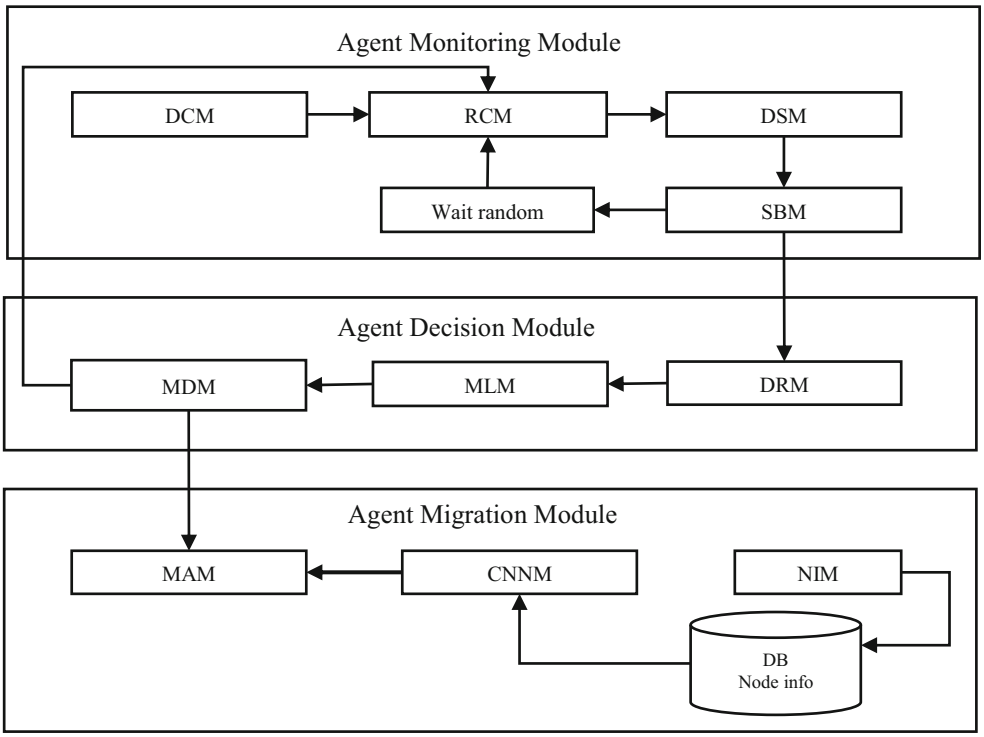

Fig. 3 Mobile agent design architecture

predict the behavior of the system providing high performance and flexibility. The component diagnosis and monitoring (CDM) systems are modular to guarantee flexibility to alter when needed in case of expansion of any module without any new line of code [3]. Localhost Information Service Agent (LISA) is a lightweight dynamic service capable of providing application monitoring, configuring system parameter and optimizing distributed applications [29]. LISA framework is independent and it can be deployed on any node architecture or operating systems. The framework (core system) is based on modules responsible for managing and monitoring other modules as well as to continuously monitor itself and detect any potential problems and take remedial measures for a solution. Precedence Based Monitoring (PBM) algorithm is proposed for load monitoring in cloud architecture to manage resources based on time, event and precedence [30]. Reduced Penalty Class Algorithm (RPCA) is used for negotiation and service level agreement between the centralized node of cloud and the consumers. In their proposed model, if the user request is not fulfilled by a particular cloud, then the agent of the current cloud migrates the request of the users to a neighboring cloud.

Centralized-But-Distributed fuzzy dynamic load balancing model is used to deal with inaccurate load information, making load distribution decision and, maintaining overall system stability [6]. This model specifies how, when and by whom load balancing mechanism is implemented. To collect the currently available status of a node, load balancer node broadcast request message to all available nodes for determining their

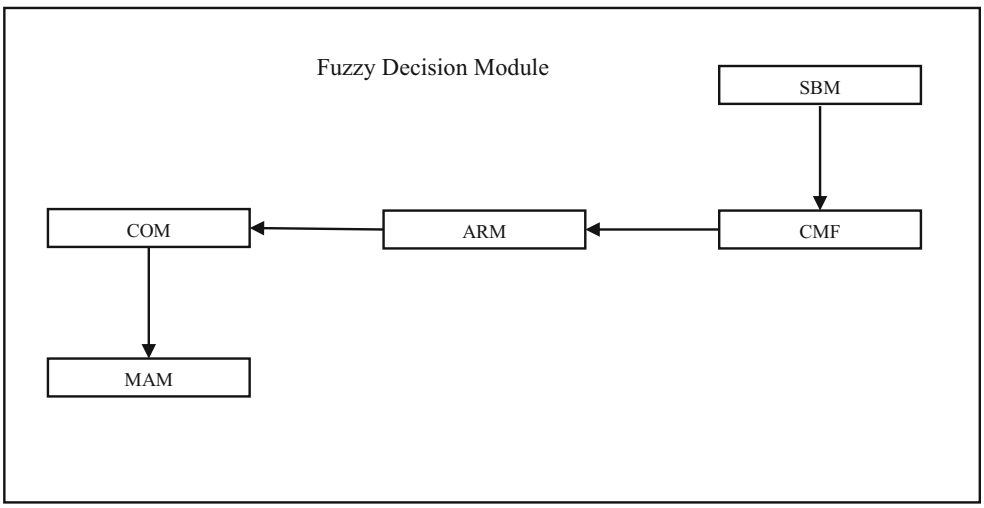

Fig. 4 Fuzzy based decision architecture module

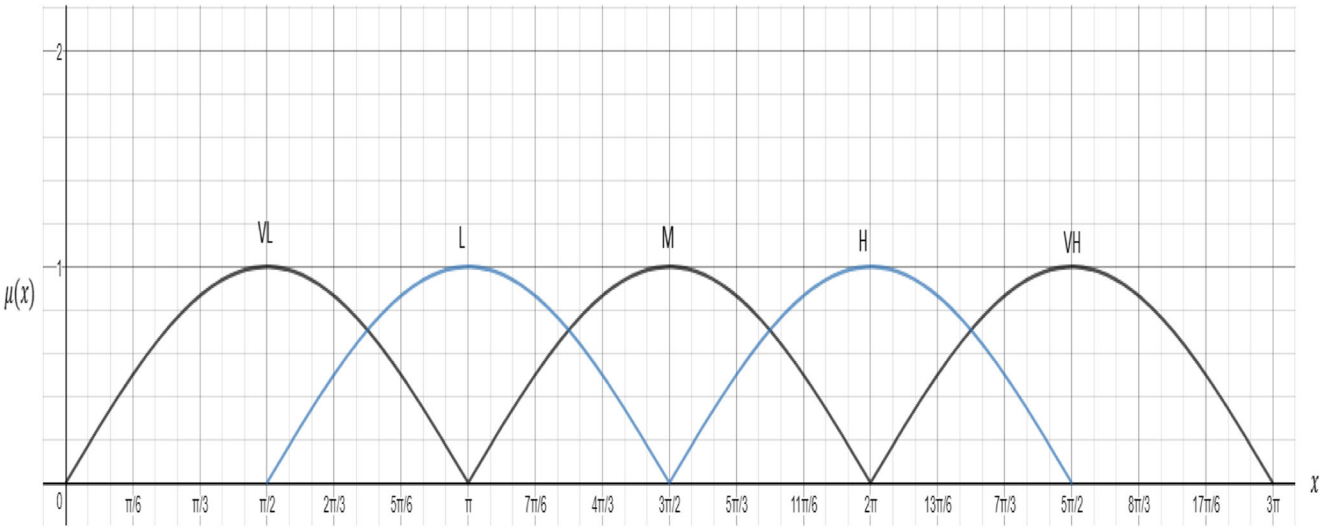

Fig. 5 Composite fuzzy membership function

current status. This model removes the single point of failure because each node can act as the load balancer node on a temporary basis at the cost of high intercommunication due to the broadcasting request message to all the connected nodes. Intelligent fuzzy grouping system (IFGS) model is designed to characterize uncertainties in the decision-making process and to decrease the messaging overhead by dividing the servers into different groups [8]. The IFGS model consists of five components: (a) load monitor, (b) load-collector, (c) fuzzy-analyzer, (d) load-indicator and, (e) load-strategy. To reduce communication overhead the network is subdivided into a small group of nodes by sending a message to its respective designated representative (DR). In the normal operations, load migration will take place if a node is leaving a subgroup. However, abnormal exit or disconnection of a node in a subgroup loses data with no backup measures. A two-level fuzzy model is designed for dynamic load balancing in cloud computing [7]. This model is used to characterize the uncertainty in a distributed system by employing fuzzy logic. This model has a hierarchical structure which consists of a cluster load manager (CRLM) and cloud load manager (CDLM). This model employs load balancing into two levels. In level 1, the CDLM will select the appropriate cluster with better processing power and with minimal queue length by employing fuzzy logic. In level 2, CRLM will select the appropriate node in a particular cluster to assign a task based on CPU load and the queue length of that node by using fuzzy logic. The researcher has proposed a dynamic load balancing model based on fuzzy logic for grid computing services [31]. This model is designed by using a fuzzy logic inference

Table 1 Fuzzy inference rule-base

	RAM				
CPU	VL	L	M	H	VH
VL	SAL	SAL	SAL	SCL	SCL
L	SCL	SCL	SCL	SCL	SCL
M	SRL	LCL	LCRL	CA	CA
H	SRL	LRL	SSL	CA	SSL
VH	SRL	LRL	SSL	SSL	SSL

SAL Send_Any_Load, *SCL* Send_Cpu_Load, *SRL* Send_Ram_Load, *LCL* Light_Cpu_Load, *LRL* Light_Ram_Load, *LCRL* Light_Cpu_Ram_Load, *SSL* Stop_Sending_Load, *CA* Compute_Again

Data types:

double mem, cpuperc, ram_per, cpu_per;
array cpu_load [], array ram_load [];
double v_1, v_2, v_3, d_{12}, d_{13}, d_{23}, v_n, v;

Initialization:

mem = null, cpuperc = null, ram_per = 0, cpu_per = 0;

Procedure:

Compute_Node_Status (cpu_load, ram_load) {
mem = get_memory_free ();
cpuperc = get_cpu_free ();
ram_per = compute_free_ram_percentage ();
cpu_per = compute_free_cpu_percentage ();
store (cpu_load [], cpu_per);
store (ram_load [], ram_per);
wait (Random());
v_1 = (cpu_load [0] . ram_load[0]);
v_2 = (cpu_load [1] . ram_load[1]);
v_3 = (cpu_load [2] . ram_load[2]);
d_{12} = | v_1 - v_2 |;
d_{13} = | v_1 - v_3 |;
d_{23} = | v_2 – v_3 |;
v_n = min (v_1, v_2, v_3);
v = |min (v_n) + ((d_{12} + d_{13} + d_{23}) /3)|;
monitoring_decision (v);}

Fig. 6 Pseudo-code representation of monitoring algorithm

```
Data types:
String v_history, v_current;
double cpu_load, ram_load;
integer msg_history;
Initialization:
cpu_load = 0, ram_load = 0, msg_history = 0;
Procedure:
//for zone_1:
monitoring_decision(){
if (cpu_load > 0.85 ∨ ram_load > 0.85){
migrate (Agent);
}elseif (v > 0 ∧ v ≤ 0.24){
v_current = zone_1;
if ((v_history == zone_1) ∨ (v_history == null) ∧ (v_current == zone_1)){
send_msg (monitoring_node, <load_any, node_id>);
v_history = v_current;
migrate (Agent);}
if (v_history == zone_2 ∧ v_current == zone_1){
compute (cpu_load, ram_load);
if (cpu_load > ram_load){
send_msg (monitoring_node, <load_light_cpu, node_id>);
v_history = v_current;
migrate (Agent);}
}else {
send_msg (monitoring_node, <load_light_ram, node_id>);
v_history = v_current;
migrate (Agent);}
if (v_history == zone_3 ∧ v_current == zone_1){
send_msg (monitoring_node, <load_light_cpu, node_id>);
v_history = v_current;
migrate (Agent);}}}
```

Fig. 7 Pseudocode representation of decision algorithm for zone_1

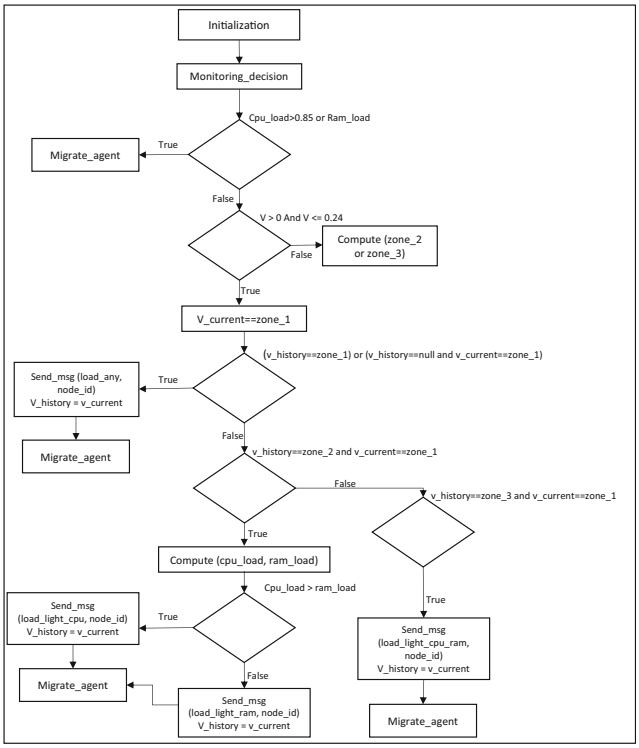

Fig. 8 Flow diagram of decision algorithm for zone_1

```
//for zone 2
elseif (v > 0.24 ∧ v ≤ 0.48){
v_current = zone_2;
if (v_history == zone_1 ∧ v_current == zone_2){
compute (cpu_load, ram_load);
if (cpu_load > ram_load){
send_msg (monitoring_node, <load_light_cpu, node_id>);
v_history = v_current;
migrate (Agent);}
else {
send_msg (monitoring_node, <load_light_ram, node_id>);
v_history = v_current;
migrate (Agent);}}
if ((v_history == zone_2) ∨ (v_history == null) ∧ (v_current == zone_2)){
if (msg_history ≥ 3){
msg_history = 0;
compute (cpu_load, ram_load);
if (cpu_load > ram_load){
send_msg (monitoring_node, <load_ram, node_id>);
v_history = v_current;
migrate (Agent);
} else {
send_msg (monitoring_node, <load_cpu, node_id>);
v_history = v_current;
migrate (Agent);}
} else {
send_msg (monitoring_node, <load_light_cpu_ram, node_id>);
v_history = v_current;
msg_history ++;
migrate (Agent);}}
if (v_history == zone_3 ∧ v_current == zone_2){
v_history = v_current;
migrate (Agent);}
}
```

Fig. 9 Pseudocode representation of decision algorithm for zone_2

system which uses specific metrics to collect the uncertainty of loads and specifies the state of each node per cluster. The load balancing mechanism is divided into two levels. Level 1: load balancing is done at the cluster level by a global manager. In Level 2: load balancing is done at the node level by a local manager. In this model, the local manager is responsible to

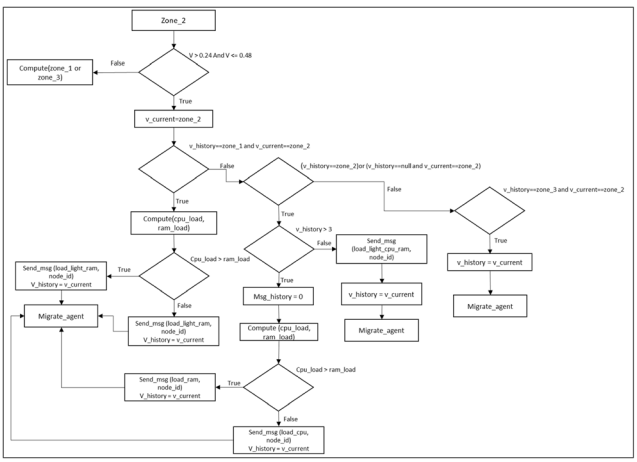

Fig. 10 Flow diagram of decision algorithm for zone_2

```
//for zone 3:
elseif (v > 0.48 ∧ v ≤ 0.72){
v_current = zone_3;
if ( v_history == zone_1 ∧ v_current == zone_3){
wait (Random());
Compute_Node_Status();
monitoring_decision();
}
if (v_history == zone_2 ∧ v_current == zone_3){
v_history = v_current;
migrate (Agent);}
if ((v_history == zone_3) ∨ (v_history == null) ∧ (v_current == zone_3)){
send_msg (monitoring_node, <stop_sending_load, node_id>);
v_history = v_current;
migrate (Agent);
}}
```

Fig. 11 Pseudocode representation of decision algorithm for zone_3

evaluate the state of the nodes inside a specific cluster as well as to perform local load balancing. The global manager will communicate with the local manager of each cluster to obtain the status of all the nodes within a cluster. The learning by observation agents (LbO model) observe the behavior performed by an expert and learn to correctly perform that behavior [32]. In this model, the agent observation of the expert is based on the quality and coverage of the behavior. LbO model uses two approaches for observation acquisition. The first one is mixed-initiative observation acquisition, while the second one is delayed observation acquisition [32]. Weighted wavelet support vector machine (WWSVM) model is used to predict the host load sequence in the cloud data center, which is helpful for resource scheduling to minimize energy consumption [33]. This model combines the advantages of wavelet transform and support vector machine in order to increase the accuracy of load prediction in cloud data center [33]. Moreover, to find the optimal combination of the parameters, the researchers have proposed a parameter optimization algorithm which is based on particle swarm optimization (PSO).

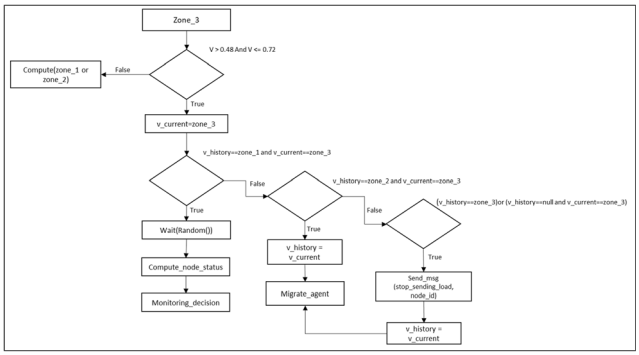

Fig. 12 Flow diagram of decision algorithm for zone_3

3 Designing monitoring agent

In this section, we have presented a high-level conceptual schematic diagram of our proposed mobile agent monitoring system as illustrated in Fig. 2. It is presented in Fig. 2, that our schematic diagram is consists of four groups of heterogeneous computing nodes. The agent body is consisting of three main modules such as (a) Agent Monitoring Module (AMM), (b) Agent Decision Module (ADM) and, (c) Agent Migration Module (AMMO). First, the AMM module collects the current status information of the available resources in random time intervals to obtain different samples. The purpose of collecting different samples is to compare the status of resources at randomized time intervals. The ADM module computes the collected samples of data for decision making at multiple levels. The AMMO module migrate the agent to appropriate nodes in distributed systems if a migration decision is made autonomously. The background processes or daemon processes are the periodic or non-terminable programs. The background processes have significance in managing system performance. In this paper, we are referring RAM and CPU as the main available system resources in a node for computational purposes.

Data types:

double mem, cpuperc, ram_per, cpu_per, condition_cpu, condition_ram, z;
array cpu_load [], array ram_load [];
double cpu_very_lightly_loaded, cpu_lightly_loaded, cpu_moderatly_loaded, cpu_highly_loaded, cpu_very_highly_loaded, ram_very_lightly_loaded, ram _lightly_loaded, ram _moderatly_loaded, ram _highly_loaded, ram _very_highly_loaded, u_output, output_rule, data_ram, data_cpu;

Initialization:

mem = null, cpuperc = null, ram_per = 0, cpu_per = 0;

Procedure:

Compute_Node_Status(){
mem = get_memory_free();
cpuperc = get_cpu_free();
ram_per = compute_free_ram_percentage();
cpu_per = compute_free_cpu_percentage();
store (cpu_load [], cpu_per);
store (ram_load [], ram_per);}
map_membership_function (cpu, ram);
Compute_Membership(){
cpu = new Cpu();
ram = new Ram ();
ram.data_ram = condition_ram;
ram_very_lightly_loaded = ram.very_lightly_loaded ();
ram_lightly_loaded = ram.lightly_loaded ();
ram_moderatly_loaded = ram.moderatly_loaded ();
ram_highly_loaded = ram.highly_loaded ();
ram_very_highly_loaded = ram.very_highly_loaded ();
cpu.data_cpu = condition_cpu;
cpu_very_lightly_loaded = cpu.very_lightly_loaded ();
cpu _lightly_loaded = cpu.lightly_loaded ();
cpu _moderatly_loaded = cpu.moderatly_loaded ();
cpu _highly_loaded = cpu.highly_loaded ();
cpu _very_highly_loaded = cpu.very_highly_loaded ();}
Apply_rule();
Compute_output(){
z = 0;
stat1 = 0;
stat2 = 0;
for (int i = 0; i < 25; i++){
stat1 += output_rule[i] * u_output[i];
stat2 += output_rule[i];}
z = stat1/stat2;}
Find_Min (double a, double b){
double result;
List < Double > list = new ArrayList <Double> (2);
list.add(a);
list.add(b);
Collection.sort(list);
result = Collection.min(list);
return result;}
RunFuzzy(){
Compute_Membership();
Apply_rule();
Compute_Output();}

Fig. 13 Pseudocode representation of Fuzzy decision algorithm

The internal architectural design of our proposed mobile agent-based monitoring system is illustrated in Fig. 3. In this section, we will be explaining the detail working of each sub-module of our proposed agent monitoring system. The AMM module consists of sub-modules as, *Data Collection Module* (DCM), *Resource Compute Module* (RCM), *Data Sampling Module* (DSM) and, *Storage Buffer Module* (SBM). The DCM module is used to collect the available status of current resources, compute the total number of processes that are currently holding the resources and sends this information to the next sub-module (RCM). The RCM module computes the received data and converts it into a percentage, which helps to efficiently determine the exact usage of available resources in the random time intervals. The DSM module selects different samples randomly and stores it in buffer storage, however, this process keeps going until a specified amount of sampling. The purpose of using a randomized waiting time is to make sure that the samples collected by the DSM module are distributed over time with relative uniformity. To avoid the effects of rapidly transiting processes in a system, the randomization of waiting time is employed. The SBM module is used to store all the sampling results which are collected by the sampling module. The sampling module stores the status information of both the RAM and CPU, then wait for the random amount of time to continue sample collection. The ADM module consists of sub-modules as, *Data Retrieval Module* (DRM), *Monitoring Logic Module* (MLM) and, *Monitoring Decision Module* (MDM). The DRM module retrieves stored data instantaneously from SBM module and forward it to MLM module for further processing.

The MLM module computes the joint probability variation of processing load in a node based on sampled data. The joint probability is used to calculate two instantaneous events that are probably occurring at the same time. By taking the joint probability of the resources, it is easy to make a decision about the status of a particular node (i.e. heavily loaded or lightly loaded) based on a suitable norm function. The MLM module will categorize the status of system load according to the embedded logic to achieve the desired goal. The MDM is used to make a decision either to migrate the agent or to re-compute the status of resources. The MDM used joint probability for decision making such that, if the joint probability of a node is greater than a threshold value, the MDM causes the agent to migrate otherwise will send the agent to RCM module for re-computing the status information. The AMM module consists of sub-module which are, *Node Info Module* (NIM), *Computing Next Node Module* (CNNM) and, *Migrate Agent Module* (MAM). The NIM module frequently checking the available nodes to determine the online and offline systems, to ensure reliability as well as increase the overall system performance by updating its database. The CNNM module select appropriate nodes based on a criteria such as, (a) select such a node with high level of processing and computing resources, (b) select a node with enough free available resources, (c) select nearest or adjacent node and, (d) a node with fast and less congested network link. The MAM module is used to migrate the agent to the next node for computing the current status of the node. The MAM module migrates the agent for computing the current status of the next node. This module uses two inputs for processing such that it received on input from the MDM module and the other input from CNNM

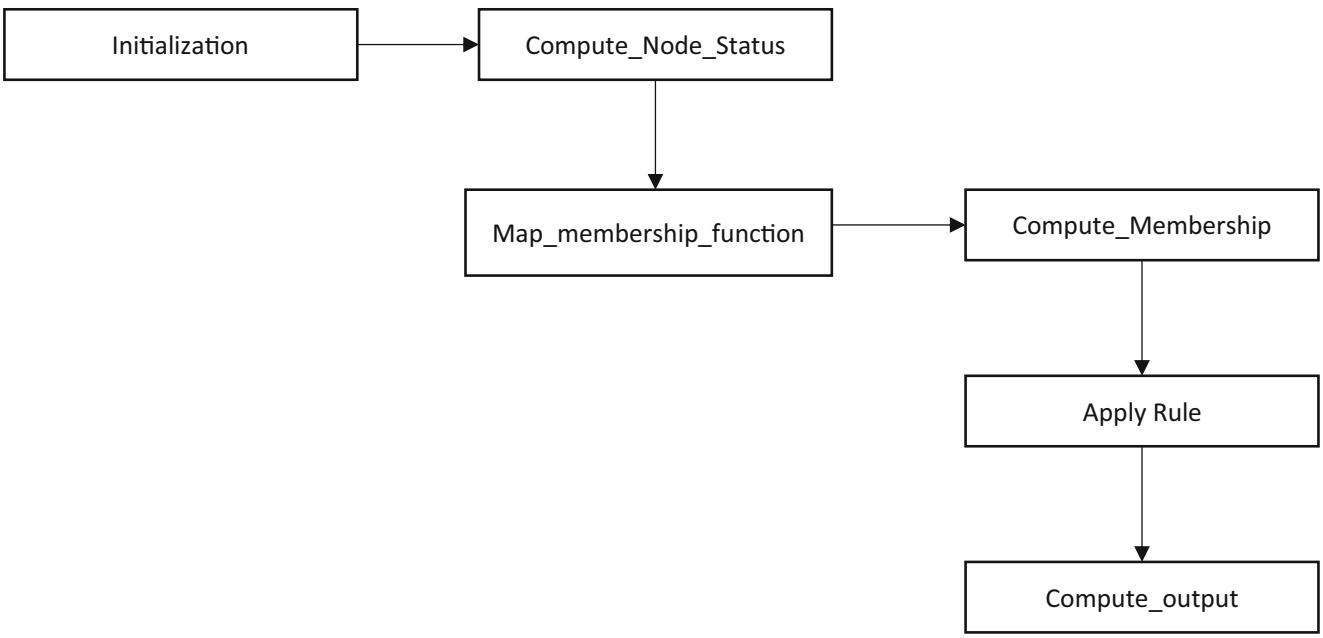

Fig. 14 Flow diagram of Fuzzy decision algorithm

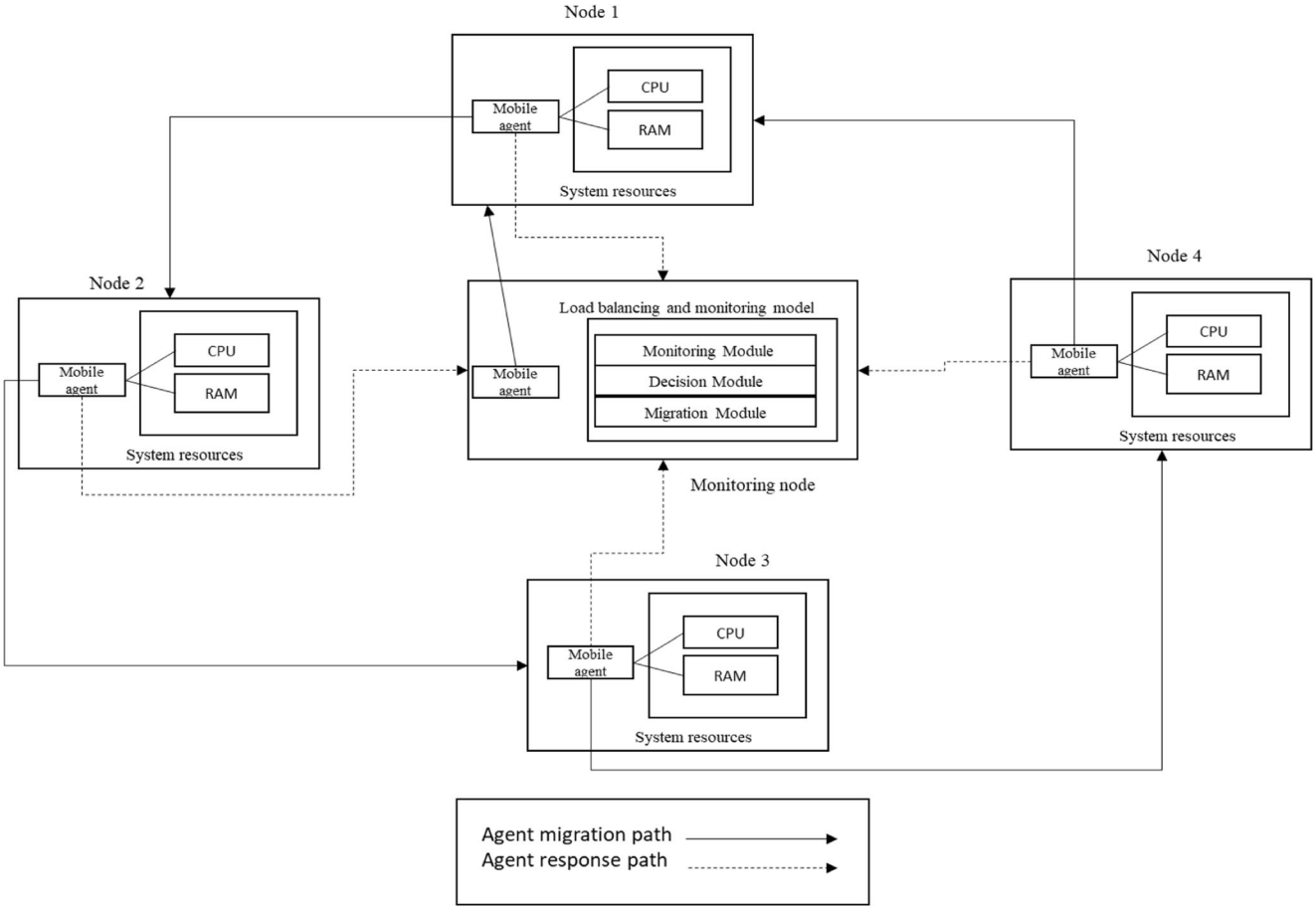

Fig. 15 Deployment model of mobile agent monitoring

module. The MDM module confirms the migration of agent, while the CNNM module will specify the node for migration.

3.1 ADM and fuzzy module hybrid

In our proposed agent-based monitoring model, ADM module is used to compute the collected data for making multiple levels of decisions. In order to increase the performance and accuracy of our monitoring model, we have removed the ADM module and added Fuzzy Decision Module (FDM). The Fuzzy Decision Module is illustrated in Fig. 4.

Fuzzy logic decision module is used to collect fine-grained information of nodes for decision-making and load monitoring in our proposed model. The FDM module consists of sub-modules

Table 2 Platform specifications of the runtime environment and system configuration

Nodes	Specification	Runtime Environment	
		Operating System	Software
Node 1	Intel Celeron G1840 CPU 2.80 GHz, RAM: 4 GB, HDD: 128 GB, NIC: Wireless Adaptor	Windows 10	Eclipse 4.6, JADE 4.5.0, JDK 1.8, CPU Stress and Heavy Load.
Node 2	Intel Core i7–6700 CPU 3.40 GHz, RAM: 8 GB, HDD: 2 TB, NIC: Realtek PCIe Controller	Linux kernel 2.6 Fedora	
Node 3	Intel Core i5 3.1GHz, RAM: 3 GB, HDD: 500 GB, NIC: Realtek PCIe Controller	Windows 10	
Node 4	Intel Core 2 Duo E8400 CPU 3.00 GHz, RAM: 3 GB, HDD: 320 GB, NIC: Realtek PCIe Controller	Windows 7	
Monitoring Node	Intel Core i7–6700 CPU 3.40 GHz, RAM: 8 GB, HDD: 2 TB, NIC: Realtek PCIe GBE Family Controller	Windows 10	
Network	Ethernet: 100Mbps LAN Wireless: 100Mbps WAP, Signal strength: 45% (average)		

for retrieving sampling data, Computing Membership Function (CMF), Apply Rules Module (ARM) and Compute Output Module (COM) for mobile agent migration. The CMF module is used to map the retrieved sampled data into appropriate membership function. We have proposed a smooth fuzzy membership function for mapping the sample data. The smooth membership function collects the fine grain information of the currently available node status. ARM module is used to apply rules to the membership values, this module is responsible to accurately categorize the membership values and apply rules accordingly. The ARM module contains twenty-five different sets of rules. The COM module retrieves data from ARM module for defuzzification. Moreover, this module computes data from the rule base and further process the result for decision-making.

3.2 Agent-based load estimation model

The aim of calculating the joint probability variation of CPU load and RAM load is to obtain the combined load status of a node n. In general, the probability based estimation model computes the uncertainity in system dynamics enabling decision making to enhance the performance and efficiency of a system under uncertain conditions [34]. Our estimation model computes the joint probability of CPU load ($P_n(C)$) and RAM load ($P_n(R)$) variations over time based on discrete samples. The set of samples represented in this experiment are consisting of randomly collected statistical datasets within an estimation time window. The aim of using this procedure is to reduce enlargement of memory consumption in a node. Moreover, this set of collected samples are enough to formulate probabilistic normed estimation of dynamics of load variations of a node. The reason for using a comparatively lower number of sampling is to avoid the repetition of the samples at various intervals generating high data volume. Moreover, this set of samples are temporarily stored in the history (kept in RAM) for decision-making purpose. Hence, in our system effectively more than three samples are taken into consideration for the decision-making process at every instant. Hence,

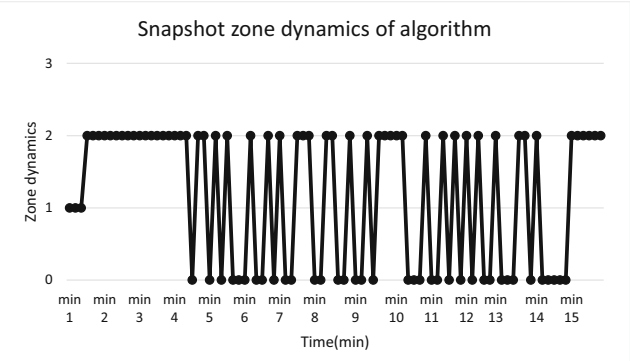

Fig. 16 Zone dynamics of mobile agent monitoring system (ex-set-1)

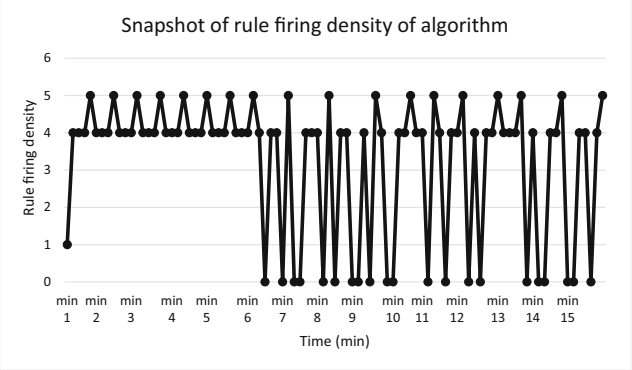

Fig. 17 Rule firing density of mobile agent monitoring system (ex-set-1)

the computation of the discrete joint probability of variations of resource load in a time window at the node n is given below.

$$t\in Z^+, t<4,\quad \wp_n(t)=\left(P_n(C)\Big|_t\right).\left(P_n(R)\Big|_t\right) \tag{1}$$

The relative triangular distances between the three samples of discrete joint probabilities are computed as,

$$a,b\in Z^+,\quad d_n(a,b)=|\wp_n(a)-\wp_n(b)| \tag{2}$$

The minimum value of joint probabilities of three different time intervals are computed as,

$$v_n=\min\{d_n(a,b):a,b<4\} \tag{3}$$

The joint resource load of the corresponding node is computed by employing a norm function as,

$$\|v\|=|v_n+avg(d_n(a,b))| \tag{4}$$

The probabilistic norm is the broad generalization of an ordinary normed linear space representing lengths of elements within the space in the non-negative range [35].

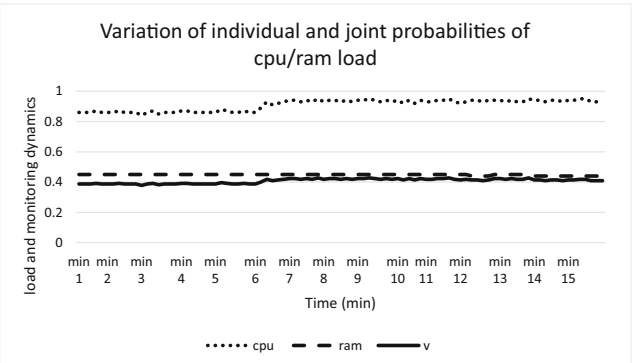

Fig. 18 Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-1)

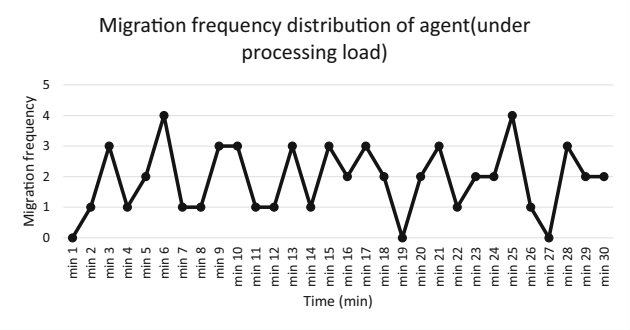

Fig. 19 Migration frequency distribution of agent under cpu_load (ex-set-1)

It is easy to verify that Eq. (4) is a norm because, $\forall t \in Z^{+}$, $|A_t| \geq 0$, $\forall k \in \Re$: $|kA_t| = |k| . |A_t|$ and, $|A_{t=a}| + |A_{t=b}| \geq |A_{t=a} + A_{t=b}|$ where, $A_t = v_n + avg(d_n(a, b))$ at a time instant t.

The norm value $\|v\|$ of a node is used as an input parameter for decision making by load monitoring algorithm of the mobile agents.

4 Designing fuzzy load balancer

In our proposed approach fuzzy logic is used for decision making under uncertainties. In our proposed model, we have formulated a smooth composite membership function. The smooth function is able to extract fine-grained information from the data and able to provide analysis that is flexible and robust [36, 37]. We have fuzzy integrated logical module into our agent-based monitoring system to deal with the uncertainty of dynamic load balancing and decision-making purpose. The composite profile of a fuzzy membership function is given in Fig. 5.

Our proposed membership function is a composite of two function sine and cosine for collecting fine-grained information of node resources. It is periodic in nature because it is repeating over fix intervals of pi radians. In our rule-based system, we have two antecedent for CPU and RAM and the linguistic variables are, very_lightly_loaded, lightly_loaded, moderately_loaded,

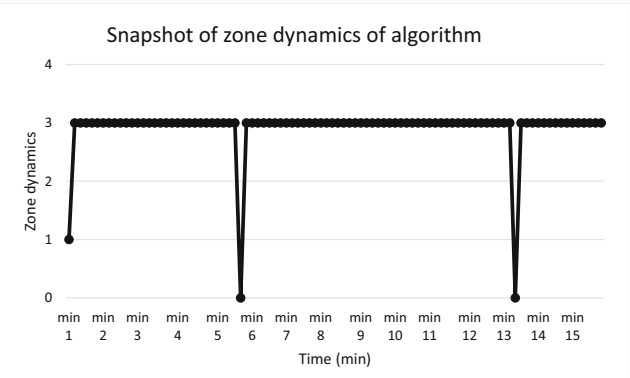

Fig. 20 Zone dynamics of mobile agent monitoring system (ex-set-2)

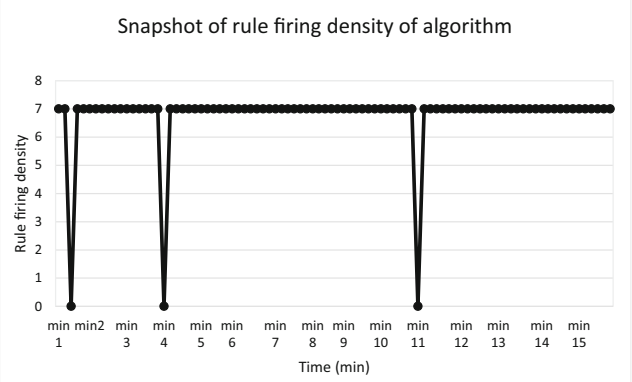

Fig. 21 Rule firing density of mobile agent monitoring system (ex-set-2)

highly_loaded and very_highly_loaded. While our consequent linguistic terms are Send_Any_Load, Send_Cpu_Load, Send_Ram_Load, Light_Cpu_Load, Light_Ram_Load, Light_Cpu_Ram_Load, Stop_Sending_Load, Compute_Again. Fuzzy reasoning (if-then) rules have two basic features. The first feature is that the rule will partially match the input data to make an inference. The second feature is that the fuzzy inference system will combine all the conclusions of the rules to form a conclusion. The rule-based system for our proposed model is illustrated in Table 1.

5 Monitoring and decision algorithms

The algorithm is designed based on the execution logic of mobile agents. We have employed a mobile agent to collect status information from nodes. The agent is capable to migrate autonomously based on decision logic. The agent sends response messages from target nodes to the monitoring node. Designing of the monitoring algorithm and decision algorithm are given below. In our proposed model, we have designed two different types of decision algorithms named, 1) Agent-based decision algorithm and, 2) fuzzy based decision algorithm. Agent-based decision algorithm uses a mobile agent to collect the status information autonomously and carried out the decision accordingly. In this model, we used probabilistic norm for calculating the status information. The fuzzy based

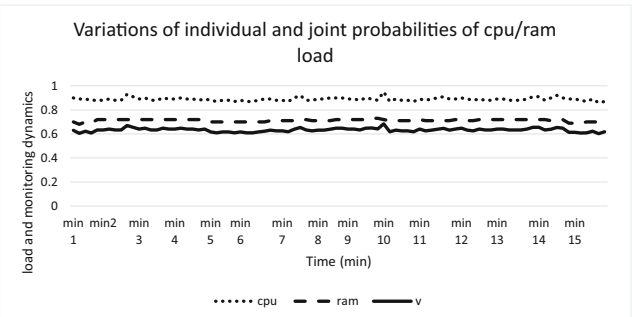

Fig. 22 Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-2)

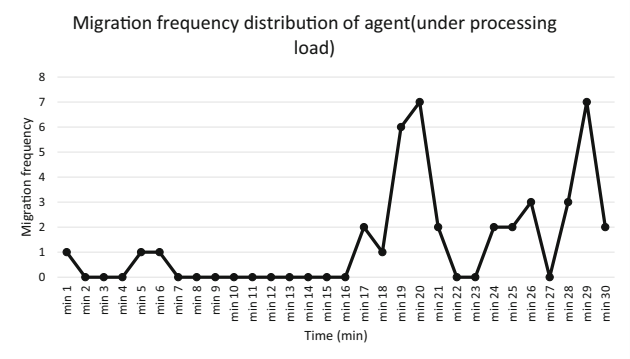

Fig. 23 Migration frequency distribution of agent under cpu_load and VOD load (ex-set-2)

decision algorithm uses fuzzy logic to make a decision based on the uncertainty of the node status. In this model, we use smooth membership function, which collects fine-grained information of node status, thus increasing the accuracy and overall performance of the system.

5.1 Agent based monitoring algorithm

The monitoring algorithm executes to compute the status of currently available resources of a node in a specified amount of time. The pseudocode representation of the monitoring algorithm is presented in Fig. 6. The monitoring algorithm computes the available resources such as cpu_load and ram_load of a node. The existing cpu_load and ram_load are computed in terms of percentage (%) to accurately determine the freely available resources of a node. In order to collect the RAM and CPU status of a node we have used two functions namely, get_memory_free () and get_cpu_free (). The returned values of cpu_load and ram_load are stored in an array for further processing. To calculate the joint probability values of v1, v2 and, v3 the data in the cpu_load and ram_load are accessed to compute the product of all the stored instances. The purpose of calculating joint probability is to determine the variations in cpu_load and ram_load in order to calculate the combined load status of a current node. The algorithm is designed to run and compute the joint probability for at least three

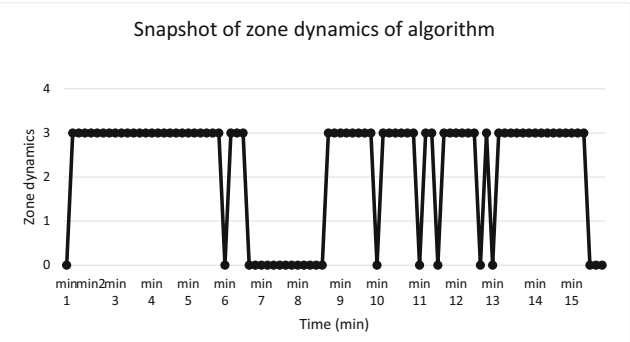

Fig. 24 Zone dynamics of mobile agent monitoring system (ex-set-3)

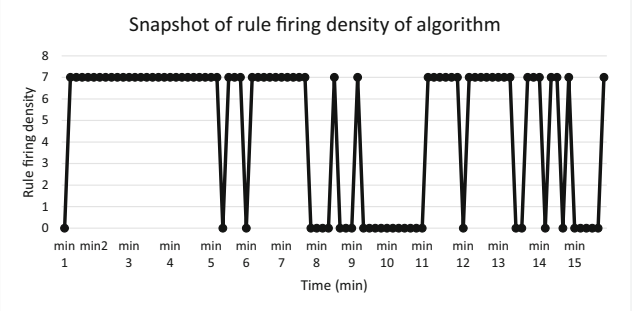

Fig. 25 Rule firing density of mobile agent monitoring system (ex-set-3)

different time instances to efficiently and effectively calculate the cpu_load and ram_load in random time intervals within an estimation time window.

The algorithm calculates the joint probabilities of resource load variations and their relative triangular distances (d_{12}, d_{13} and, d_{23}). The minimum value of joint probabilities is computed and is stored in variable v_n. Finally, the algorithm calculates the final cpu_load and ram_load of a node by following the norm function. The computed final value v (i.e. norm value) is passed as an input parameter to monitoring decision algorithm.

5.2 Agent based decision algorithms

The function of the decision algorithm is to process the input data from the monitoring algorithm for decision making. The input data processing in the decision algorithm is done in three distinct zones for current system load categorization such as zone_1, zone_2, and zone_3. Moreover, zone_1 represent the low load, zone_2 represent the medium load, and zone_3 represent high load of the current available node. The process of zone quantization is performed to determine the current load status of a node for the effective and efficient load monitoring system. The zone boundaries are set in a scale between 0 and 1, whereas 0 represent the lowest value and 1 is the highest value. The distribution of zones is based on joint probability

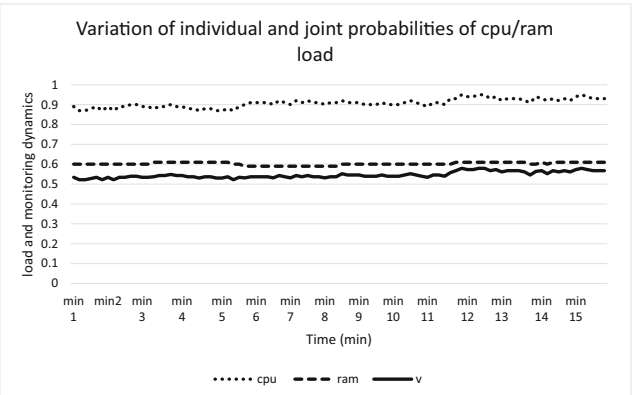

Fig. 26 Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-3)

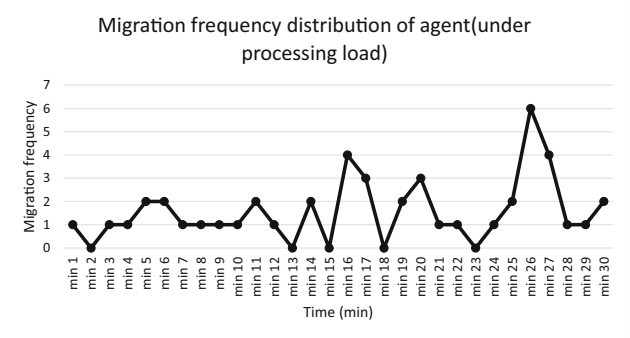

Fig. 27 Migration frequency distribution of agent under cpu_load and VOD load (ex-set-3)

values assigned to each zone. The zone boundaries are heuristically set as, $zone\_1 \Rightarrow (v > 0 \wedge v \leq 0.24)$,$zone\_2 \Rightarrow (v > 0.24 \wedge v \leq 0.48)$, and $zone\_3 \Rightarrow (v > 0.48 \wedge v \leq 0.72)$. We have divided the decision algorithm into three parts based on the current status of zones for easy understanding and description.

5.2.1 Decision algorithm for Zone_1

In our proposed decision algorithm there are two important variables named *v_history* and *v_current*. The *v_history* stores last updated zone information to capture zone dynamics and *v_current* stores current zone status. This will result in changing the status of *v_history* to *v_current* in the next execution instant. The pseudo code and flow diagram representation of the decision algorithm for zone_1 is presented in Fig. 7 and Fig. 8 respectively.

In zone_1, first the monitoring_decision function checks the cpu_load and ram_load of a node. If any of individual load estimation is greater than 0.85, then it will result in migration of agent without sending any message to the monitoring node. The *elseif* condition checks the joint probability value to identify the current zone status of a node and to inform the monitoring node for a particular load migration. The decision algorithm checks the status history (*v_history*) and, if the history is "null" or "zone_1" then the mobile agent will send a message to monitoring node indicating that this node is lightly loaded enabling to send any processing load to it. Next, the

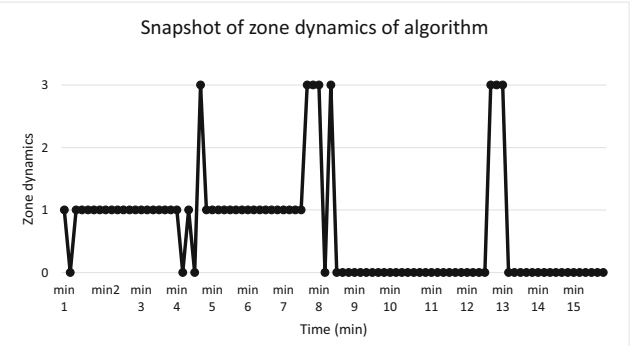

Fig. 28 Zone dynamics of mobile agent monitoring system (ex-set-4)

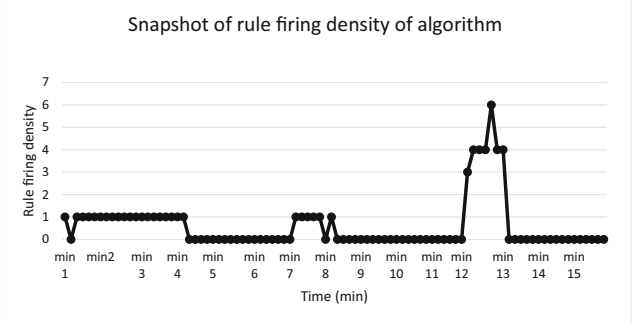

Fig. 29 Rule firing density of mobile agent monitoring system (ex-set-4)

mobile agent updates the history to zone_1 and will migrate to the next node. The second condition will check if the *v_history* is in zone_2 or not. Accordingly, the algorithm will compute the cpu_load and ram_load once more. If the cpu_load is greater than ram_load, then the mobile agent will send a message to the monitoring node for transferring light cpu_load and vice versa. Next, it will update zone dynamics history and will migrate to the next node. The third condition is to check whether *v_history* contains value zone_3 or any other zone. If it is in zone_3, then it means that this node has previously highly loaded and is changed its status to lightly loaded. As a result, the mobile agent sends a message to a monitoring node for transferring the light processing load to the target node. Next, it updates *v_history* of the node and migrates to next node.

5.2.2 Decision algorithm for Zone_2

In continuation of the first part, the second part of the decision algorithm identifies zone_2, which signifies a moderately loaded zone. The pseudo code and flow diagram representation of zone_2 of decision algorithm is represented in Fig. 9 and Fig. 10 respectively.

According to the computed joint probability value, it will assign zone_2 to a *v_current* variable.

The algorithm will check the zone history and current zone status to make further decision. In the next step, the agent will compute the cpu_load and ram_load status of the node. In case

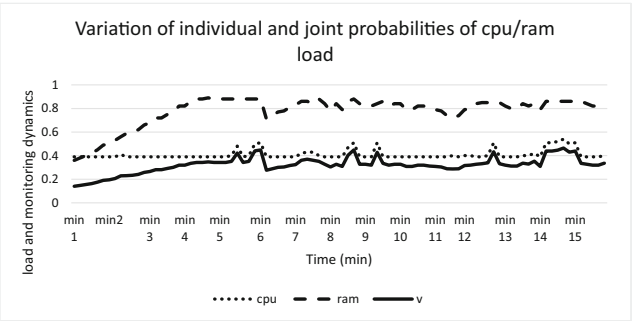

Fig. 30 Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-4)

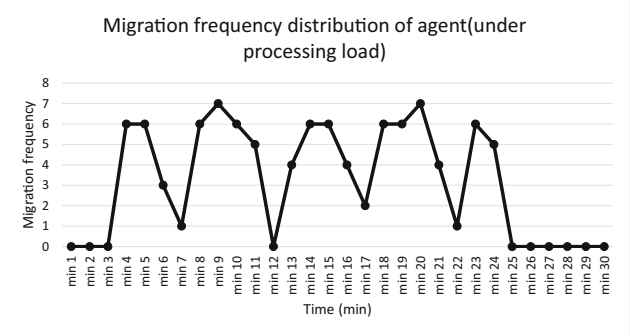

Fig. 31 Migration frequency distribution of agent under cpu_load and VOD load (ex-set-4)

the cpu_load is greater than ram_load, the mobile agent will send a message to the monitoring node to send a process with light cpu_load to this particular node. The algorithm will update the *v_history* variable and will migrate to the next node. Otherwise, when the ram_load is greater than cpu_load, the mobile agent sends a message to the monitoring node to send a process with light ram_load. The algorithm will update the *v_history* and will migrate to the next node. Based on available condition and zone dynamics, the algorithm will check the message history (*msg_history*). The message history plays a crucial role to avoid sending of the same message infinitely to monitoring node. If the message history is greater than a predefined value (which is set to three in this case), then the algorithm will compute the cpu_load and ram_load of the node once again. If the cpu_load is high then the agent sends a message to the monitoring node to transfer processes with high ram_load, otherwise, the agent sends a message to transfer processes with cpu_load. Next, the algorithm updates the *v_history* variable and initiates migration of agent. If the message history is less than a predefined value, then the agent will send a message to the monitoring node to transfer light cpu_load and ram_load to a particular node. Once the message is sent, the *v_history* variable is updated and the message variable is incremented. Lastly, the agent is migrated to the next node. If the *v_history* variable contains zone_3 and *v_current* variable contains zone_2, then the algorithm will update the *v_history* and will migrate the agent. The reason

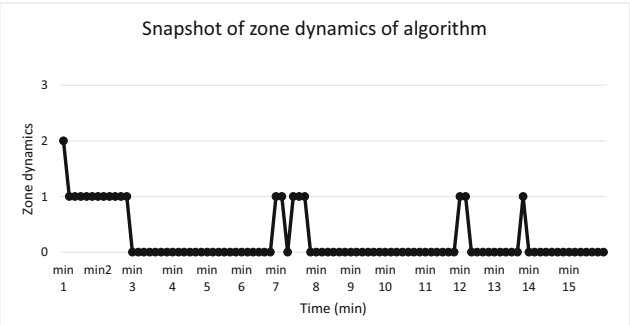

Fig. 32 Zone dynamics of mobile agent monitoring system (ex-set-5)

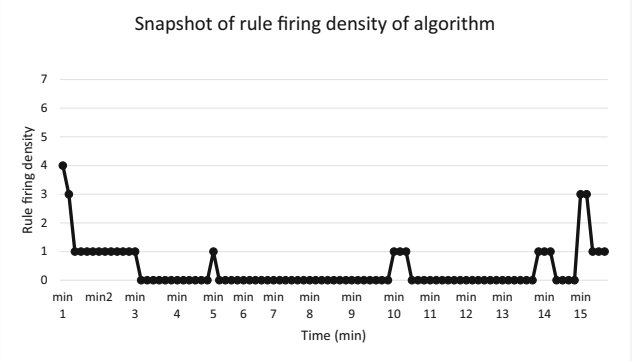

Fig. 33 Rule firing density of mobile agent monitoring system (ex-set-5)

for not taking any action by the monitoring node is to avoid an already moderately loaded node from getting a burst of processing load within a short time interval.

5.2.3 Decision algorithm for Zone_3

The third part of the decision algorithm describes the dynamics of zone_3, which signifies a highly loaded zone. The pseudo-code and flow diagram representation of zone_3 of decision algorithm are presented in Fig. 11 and Fig. 12 respectively.

According to the joint probability value, the algorithm will assign zone_3 to a *v_current* variable. If in a node, the *v_history* variable contains zone_1 and *v_current* variable contain zone_3, then the algorithm will wait for a random amount of time to repeat the estimation. The logic of using the random amount of time is to probabilistically avoid the running processes near to completion to be included within the estimation of the load. When the waiting time is over, the algorithm will recall *Compute_Node_Status* function and *monitoring_decision* function to evaluate the current zone status. If the *v_history* contains zone_2 and *v_current* contains zone_3, then the algorithm will update the *v_history* variable and will enforce agent migration without sending a message to the monitoring node. The reason for not sending a message for

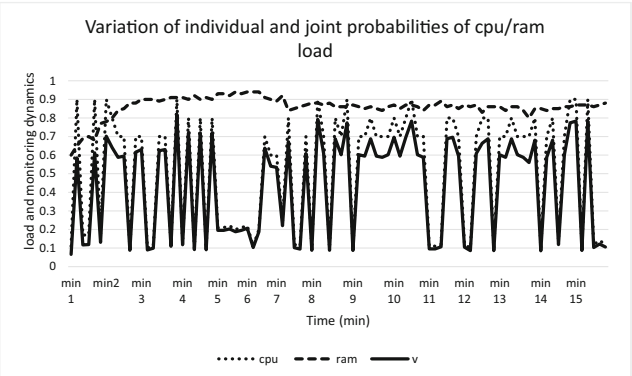

Fig. 34 Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-5)

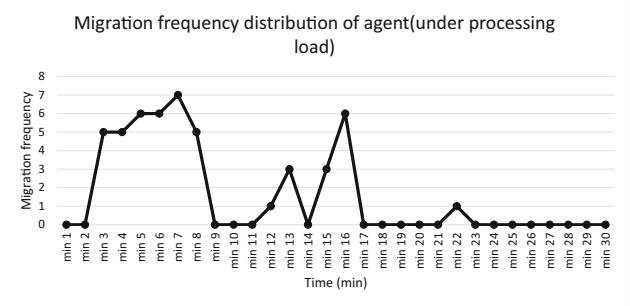

Fig. 35 Migration frequency distribution of agent under cpu_load and VOD load (ex-set-5)

transferring extra load is to avoid overloading scenario. Finally, if the *v_history* variable contains zone_3 or null value and, the *v_current* variable contains zone_3, then the agent will send a message to monitoring node to avoid transferring the load to an already highly overloaded node. After sending the message to monitoring node *v_history* variable is updated and the agent is migrated to next node.

5.2.4 Fuzzy based decision algorithm

The function of the fuzzy decision algorithm is to process the input data from the agent-monitoring algorithm for decision-making. The pseudo code and flow diagram representation of a fuzzy decision algorithm is illustrated in Figs. 13 and 14.

In this algorithm, the Computing_Node_Status function collects the node status and the computed result is stored accordingly. This function will determine the exact status of the resources instantaneously at a particular time. The Map_membership_function is used to map the values to appropriate membership variables and determine their boundaries. To calculate the membership degree of crisp values for CPU, RAM and stored it in a variable named condition_cpu and condition_ram by calling the Compute_Membership function as described in the algorithm. There are 25 rules that have been created and implemented in the program as a procedure Apply_rule as shown in the algorithm. The

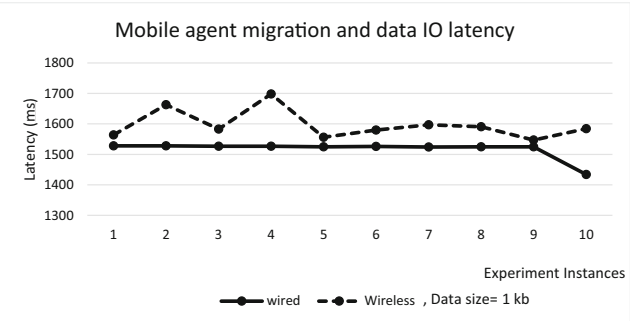

Fig. 36 Comparison of mobile agent migration and data IO latency with respect to wired and wireless node (data size = 1 kb)

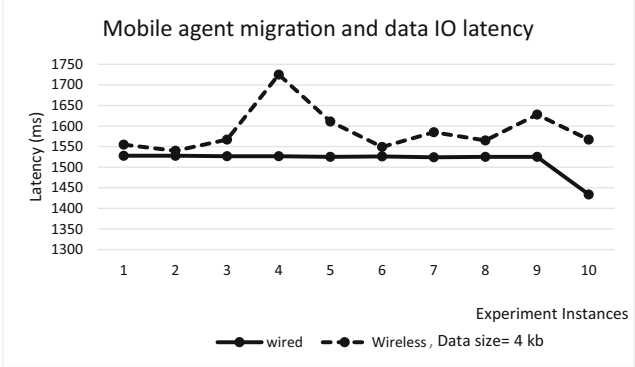

Fig. 37 Comparison of mobile agent migration and data IO latency with respect to wired and wireless node (data size = 4 kb)

Compute_output function will compute the output result, which is obtained by calculating the fired rule. The rules are fired according to the currently available resource status of a node. Implementation of the AND rule in the program is obtained by the minimum value of the membership degree of CPU and RAM. For that purpose, the Find_Min procedure is created to get the minimum value in an array. A RunFuzzy procedure is then generated to invoke a fuzzy procedure that has been previously made in accordance with the steps in fuzzy logic as shown in the algorithm.

6 Implementation environment

6.1 Agent deployment and architecture

In this section, we have described our proposed software architecture and deployment model. Our testbed is comprised of five nodes with one designated node for monitoring and the remaining five are for executing mobile agents as illustrated in Fig. 15.

The mobile agent checks the availability of a node for migration if alive, otherwise will check for the next alive node. Information related to nodes availability are continuously

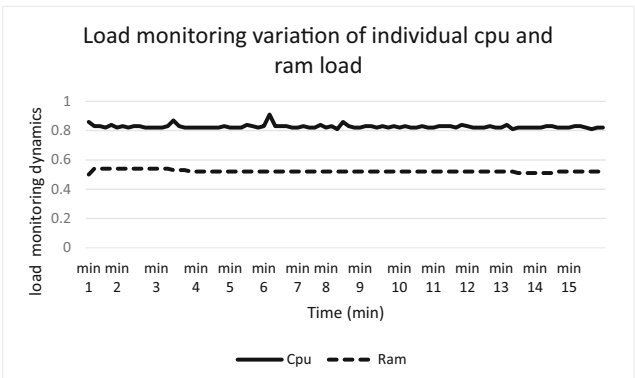

Fig. 38 Variation of load monitoring dynamics of algorithm (ex-set-1)

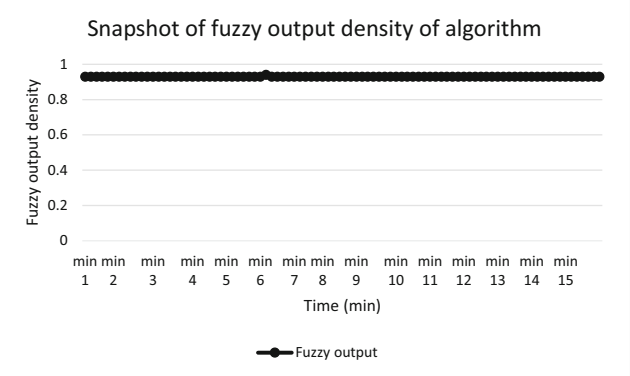

Fig. 39 Fuzzy output density of integrated module of monitoring algorithm (ex-set-1)

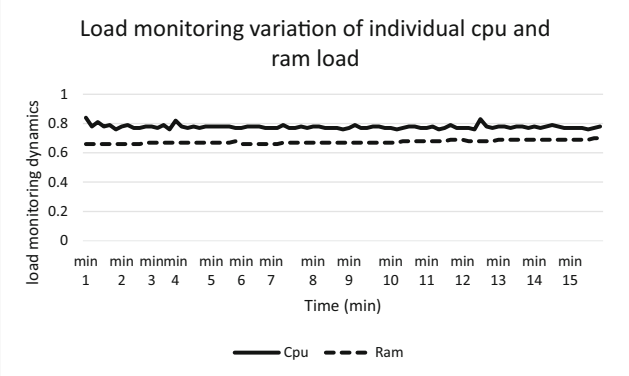

Fig. 41 Variation of load monitoring dynamics of algorithm (ex-set-2)

updated in a database. The feature of our proposed model is to enable mobile agents to collect the load status of the individual node and send the required information to the monitoring node. This mechanism will reduce the response time improving the overall system performance. On the other hand, in traditional load monitoring the current load values are collected as a whole and then forwarded to the monitoring node. The testbed for implementing our model is consists of a heterogeneous OS and Java programming language is used to deploy a mobile agent framework (i.e. Java Development Framework (JADE)). Agents are termed as containers in JADE and are installed on all the available nodes. Furthermore, the configuration of our mobile agent framework development and runtime environment specification is illustrated in Table 2. Our proposed mobile agent-monitoring algorithm for monitoring distributed system is developed using Java eclipse IDE on top of the JADE agent platform. In addition, to test our proposed monitoring algorithm, we have used two additional load generation (benchmark) software modules, which are, (a) *CPU Stress* and (b) *Heavy Load*. The purpose of using these software is to generate various categories of load on our target nodes to monitor variations in our algorithmic behavior under different load conditions. In terms of network connectivity, one of the node is wirelessly connected and the rest is wired connected with the monitoring node. The wired connections operate at 100Mbps and wireless network operates on 100Mbps at maximum.

The zonal decision-making algorithms are implemented as separate functions within the agent body (code). The total lines of code (LOC) for each agent are equal to 182.

7 Experimental evaluations

In this section, we have evaluated the performance and behavior of our proposed models under different load conditions. The set of parameters for evaluating our proposed Agent-based monitoring model is consisting of zone dynamics, rule firing density, load dynamics and, migration frequency as described in details in Section 7.1. To evaluate the behavior and performance of our second proposed fuzzy integrated model the set of parameters include load dynamics, fuzzy output density and, rule firing density as illustrated in Section 7.2.

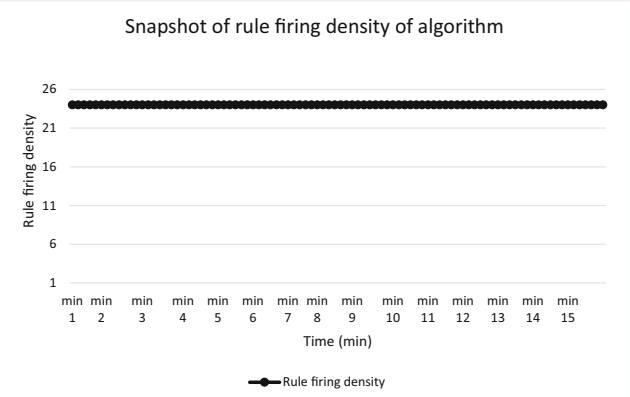

Fig. 40 Rule firing density variations of integrated module of monitoring algorithm (ex-set-1)

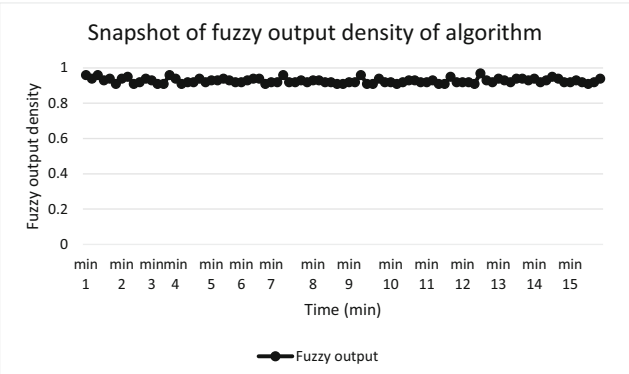

Fig. 42 Fuzzy output density of integrated module of monitoring algorithm (ex-set-2)

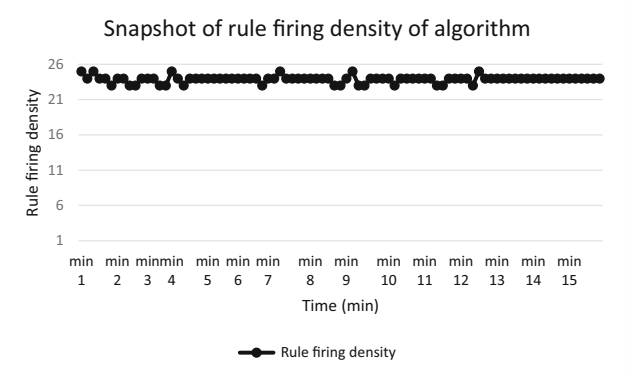

Fig. 43 Rule firing density variations of integrated module of monitoring algorithm (ex-set-2)

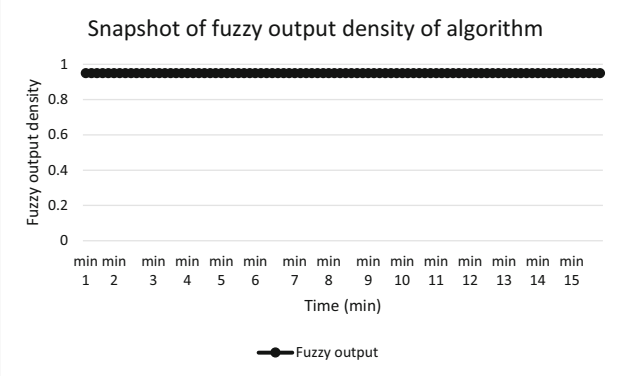

Fig. 45 Fuzzy output density of integrated module of monitoring algorithm (ex-set-3)

7.1 Evaluation of agent-based monitoring model

In this section, we are evaluating the behavior and performance of our implemented agent-based monitoring model. In set 1, the cpu_load is evaluated by applying two synthetic load generation software's named heavy load and stress benchmark, which impose a series of non-uniform stress-load profiles on the node. In set 2, the cpu_load is evaluated by applying only VOD benchmark. In set 3, the cpu_load is evaluated by applying VOD, Heavy load and CPU stress benchmark. In set 4, the cpu_load and ram_load is evaluated by employing VOD, heavy load and CPU stress benchmark. In set 5, the ram_load is evaluated by applying heavy load and VOD software.

7.1.1 Set 1: Agent-based monitoring model employing heavy load and cpu stress

The variations in zone dynamics with respect to increasing cpu_load is illustrated in Fig. 16. Initially, the CPU load is in zone_1, which is transferred to zone_2 with further increased in load. However, further increase in the CPU load will exceed a threshold value causing migration of the load. As illustrated in Fig. 17, the rule firing density resides in zone_4 with some variations to other zones with respect to CPU load. As illustrated in Fig. 18, the behavior of our proposed algorithm is observed for the combined dynamics of joint probability by a gradual increase in the CPU load and keeping RAM load at a constant rate. The variations in migration frequency are illustrated in Fig. 19.

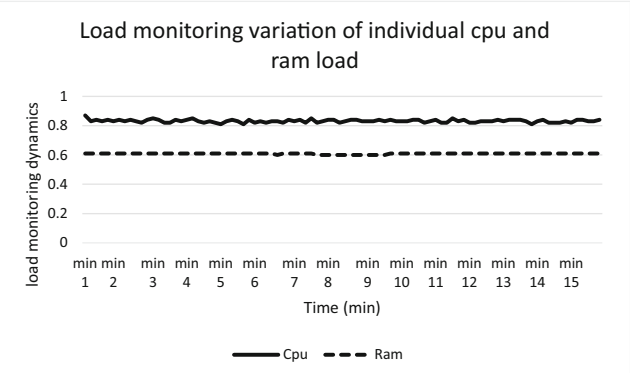

Fig. 44 Variation of load monitoring dynamics of algorithm (ex-set-3)

7.1.2 Set 2: Agent-based monitoring model employing heavy load and VOD

The variation in zone dynamics with respect to increasing cpu_load and VOD is illustrated in Fig. 20. Initially, cpu_load is in zone_1, however, a gradual increase in the load results in a transition of the node to zone_3 and it stays there without migration. However, due to background processes, an abrupt increase in the load happens exceeding the specified threshold value. This results in agent migration. The variations in rule firing density are illustrated in Fig. 21. The variations in the combined dynamics of CPU/RAM utilization and joint probability are illustrated in Fig. 22. Under increased cpu_load and VOD, the status of CPU utilization becomes maximum. The variations in migration frequency are illustrated in Fig. 23. It is shown in the figure that increasing the CPU load and VOD causes frequent migration of agent between different zones.

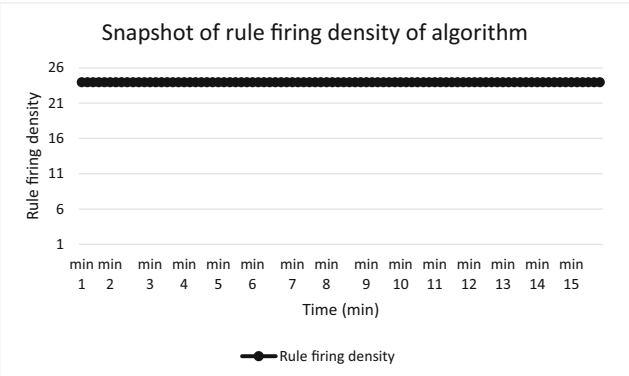

Fig. 46 Rule firing density variations of integrated module of monitoring algorithm (ex-set-3)

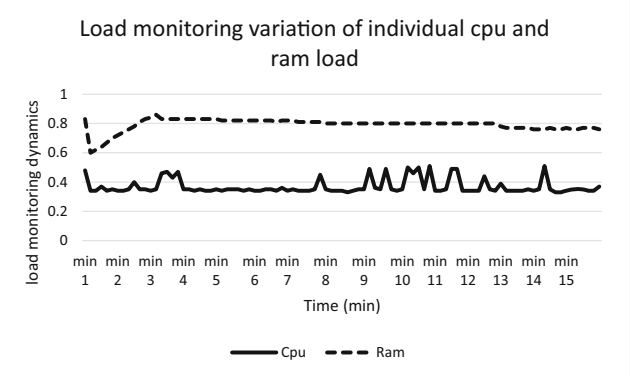

Fig. 47 Variation of load monitoring dynamics of algorithm (ex-set-4)

7.1.3 Set 3: Agent-based monitoring model employing heavy load, VOD and CPU stress

As illustrated in Fig. 24, variation in zone dynamics is observed with the initial load in zone_3. However, further increases in the cpu_load as well as in the background processes will migrate the load. As illustrated in Fig. 25, variations in rule firing density is observed with high load transfer in zone_7. However, crossing threshold values will cause the load to migrate. The variations in the combined dynamics of CPU/RAM utilization and the joint probability is illustrated in Fig. 26. The variations in migration frequency are illustrated in Fig. 27. It is shown in the figure that increasing the CPU load, VOD and cpu_stress causes frequent migration of agent.

7.1.4 Set 4: Agent-based monitoring model employing heavy load, VOD and CPU stress

The variation in zone dynamics with respect to increasing cpu_load is illustrated in Fig. 28. It observed that initially, a load of a particular node is in zone_1, which is migrated when exceeding a threshold value. The variations in rule firing density are illustrated in Fig. 29. As shown in the figure, increasing the load will cause agent migration conditioned to rule firing density. The variations in the combined dynamics of

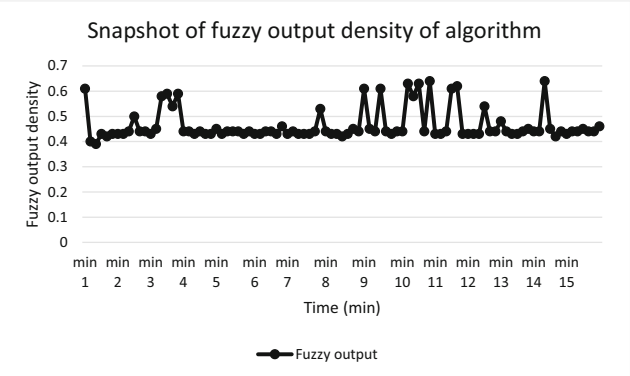

Fig. 48 Fuzzy output density of integrated module of monitoring algorithm (ex-set-4)

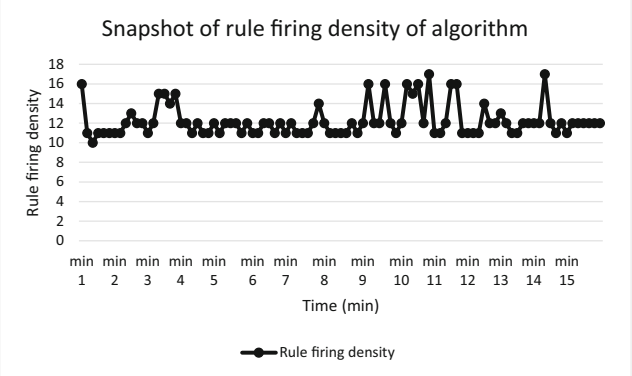

Fig. 49 Rule firing density variations of integrated module of monitoring algorithm (ex-set-4)

CPU/RAM utilization and, computed joint probability are illustrated in Fig. 30. The continuous increase in the ram_load causes the status of RAM utilization at maximum. The variations in migration frequency are illustrated in Fig. 31. Gradual increasing RAM and CPU load cause frequent migration of the agent.

7.1.5 Set 5: Agent-based monitoring model employing heavy load and VOD

The variation in zone dynamics with respect to increasing ram_load is illustrated in Fig. 32. Initially, the load is in zone_1, but a gradual increase in the ram load will exceed a threshold value to migrate the load. The variations in rule firing density are illustrated in Fig. 33. In the case of rule firing density. initially, the agent is in zone_1, but with further increase in RAM load causes the agent to migrate. The variations in the combined dynamics of CPU/RAM utilization and, joint probability are illustrated in Fig. 34. In this experiment, the Ram load is in increasing state and the CPU load is constant causes the status of RAM utilization at maximum. The variations in migration frequency are illustrated in Fig. 35. It is shown in the figure that increasing the RAM load and VOD causes frequent migration of agent between different zones.

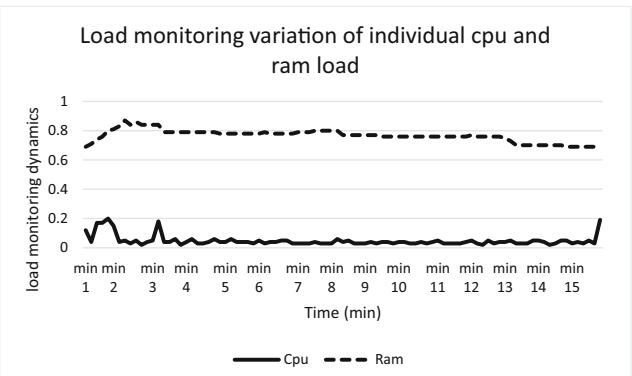

Fig. 50 Variation of load monitoring dynamics of algorithm (ex-set-5)

7.1.6 Mobile agent migration latency

In Figs. 36 and 37, the comparison of mobile agent migration latency is illustrated for the wired and wireless nodes. As illustrated in Figs. 36 and 37, the agent behavior in the wired node is relatively stable without major increase or decrease in latency, because the wired connection is more reliable and provides a stable connection to support bandwidth for a large volume of data. While in Figs. 36 and 37, the behavior of wireless node is aperiodic and tends to overshoot and undershoot with respect to network load and congestion of wireless network increasing the agent migration latency in the wireless network. The decreased latency values at instances 1, 3, 5 and 9, which is almost close to the wired node that at particular instances due to less congestion (packet loss is almost none and signal strength is good). The mobile agent migration latency is not greatly affected by the data volume it carries in the network.

7.2 Integrated evaluation

In this section, we have evaluated the agent-based monitoring algorithm integrated with fuzzy based decision-making module. The integrated module executes computing the status of currently available node resources in a specific amount of time. To evaluate the behavior and performance of our implemented model, we have conducted five sets of experiments (ex-set-1 to ex-set-5) under different load conditions. The parameter used in the evaluation of the performance are load dynamics, fuzzy output density, and rule firing density. To measure the fuzzy output density it is the total number of defuzzified output values for decision-making in a specific time interval. The function of fuzzy output density is to collect the current status of system resources to compute the defuzzified output values for decision-making. To measure the rule firing density is to calculate the total number of rules fired in given time intervals to map the input data values. The function of the rule firing mechanism is to partially match the input to make an inference and to combine all the conclusions of the rules to form a final output.

7.2.1 Set 1: Fuzzy integrated model employing heavy load and CPU stress

In the first experiment, CPU load is employed by heavy load and CPU stress benchmark for estimating the behavior of CPU and RAM load variation, fuzzy output density, and rule firing density of fuzzy integrated model. The variation in load status with respect to increasing CPU load is illustrated in Fig. 38. The variation in fuzzy output is illustrated in Fig. 39. The rule firing density variation is illustrated in Fig. 40.

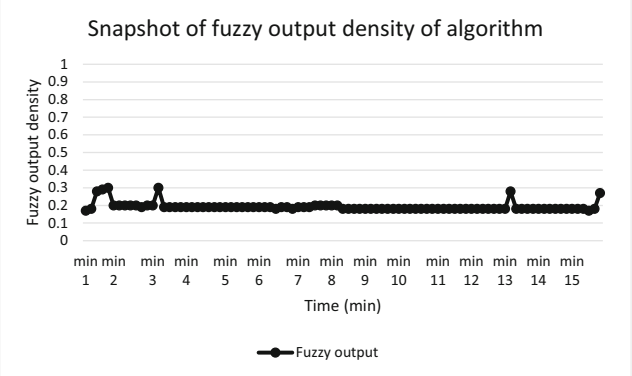

Fig. 51 Fuzzy output density of integrated module of monitoring algorithm (ex-set-5)

7.2.2 Set 2: Fuzzy integrated model employing heavy load and VOD

In this case, the CPU load is employed by using heavy load and video on demand (VOD) benchmarks for estimating the behavior of CPU and RAM load variation, fuzzy output density and, rule firing density of fuzzy integrated model. The variation of CPU and RAM status with respect to increasing CPU load and VOD is illustrated in Fig. 41. Under increased CPU load and VOD, the status of CPU utilization becomes maximum. The variation of fuzzy output density is illustrated in Fig. 42. It is shown in the figure that increasing the CPU and VOD causes the high density of fuzzy output. The variation in rule firing density is illustrated in Fig. 43.

7.2.3 Set 3: Fuzzy integrated model employing heavy load, VOD and CPU stress

In this case, the CPU load is employed by heavy load generation software, VOD, and CPU Stress benchmarks for estimating the behavior of CPU and RAM load variation, fuzzy output density, and rule firing density of fuzzy integrated model. The variation of CPU and RAM load status with respect to increasing CPU load is illustrated in Fig. 44. The variation of

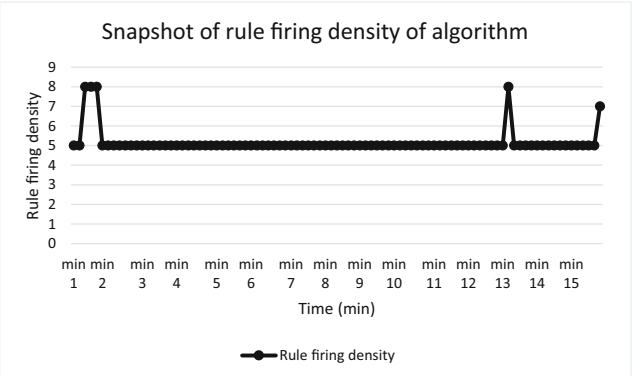

Fig. 52 Rule firing density variations of integrated module of monitoring algorithm (ex-set-5)

Table 3 Comparative analysis of mobile agent-based monitoring system and fuzzy integrated model

Parameters Monitoring models	Autonomous	Reliability	Mobility	Network latency	Accuracy	Efficiency
MABMS	H	H	H	L	M	M
FIM	H	H	H	L	H	H

H High, *M* Medium, *L* Low, Mobile Agent-Based Monitoring System (MABMS), Fuzzy Integrated Model (FIM)

fuzzy output density is illustrated in Fig. 45. The variation in rule firing density is illustrated in Fig. 46.

7.2.4 Set 4: Fuzzy integrated model employing heavy load, VOD and CPU stress

In this case, the CPU load and RAM load are employed by applying heavy load and CPU Stress benchmarks for estimating the behavior of CPU and RAM load variation, fuzzy output density and, rule firing density of fuzzy integrated model. The variation of CPU and RAM load status with respect to increasing CPU load is illustrated in Fig. 47. The variation of fuzzy output density is illustrated in Fig. 48. The variation in rule firing density is illustrated in Fig. 49.

7.2.5 Set 5: Fuzzy integrated model employing heavy load and VOD

In the last experiment, the CPU and RAM load are employed by heavy load generation software and VOD benchmarks for estimating the behavior of CPU and RAM load variation, fuzzy output density, and rule firing density of fuzzy integrated model. The variation of CPU and RAM load status with respect to increasing CPU load is illustrated in Fig. 50. The variation of fuzzy output density is illustrated in Fig. 51. The variation in rule firing density is illustrated in Fig. 52.

8 Comparative analysis

We have evaluated our mobile agent-based load monitoring model (MABMS) with various agent-based load monitoring frameworks as well as with our fuzzy integrated model (FIM) for performance analysis. In this section, we have compared our two models, agent-based load monitoring and fuzzy integrated model qualitatively as well as quantitatively. Some of the other models which we compare are, *Agent-Based Adaptive Monitoring System* [38], *Java Based Agent Management System* [39], *Localhost Information Service Agent* (LISA) [29], *Web Server Load Monitoring System using a Mobile Agent* [28], *A Method of Network Monitoring by Mobile Agents* [11], *A Mobile Agent-Based System for Server Resource Monitoring* [14], *Mobile Agent-Based Load Monitoring* [40], and *Monitoring Agents in A Large Integrated Services Architecture* (MonALISA) [41]. Each mobile agent model is examined and evaluated with respect to various design parameters and attributes like autonomous, reliability, mobility, network latency, heterogeneity and, flexibility. A detailed discussion is explained in the following sections.

8.1 Qualitative analysis of MABMS and FIM models

In this section, for the comparison purpose, we have selected six parameters for our analysis namely, autonomy, reliability, mobility, network latency, accuracy, and efficiency as illustrated in Table 3. The values of performance metric are determined with approximation by analysis. The metric values are set on a scale between 0 and 1, where 0 represents the lowest value, and 1 represents the highest value. We have divided the interval into three zones such as the first zone is from 0 to 0.3 (low zone), the second zone is from 0.3 to 0.6 (medium zone), and the last zone is from 0.6 to 1 (high zone).

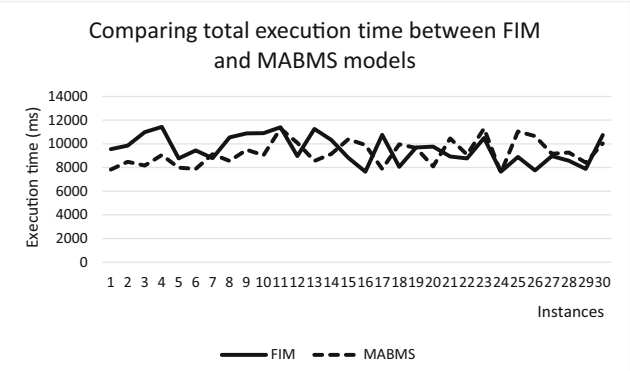

Fig. 53 Total execution time of FIM and MABMS models

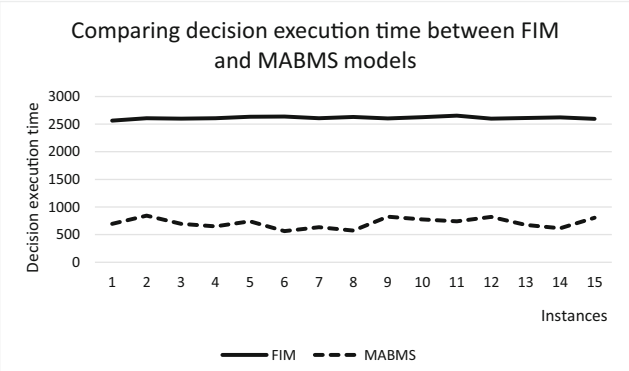

Fig. 54 Decision time of FIM and MABMS models

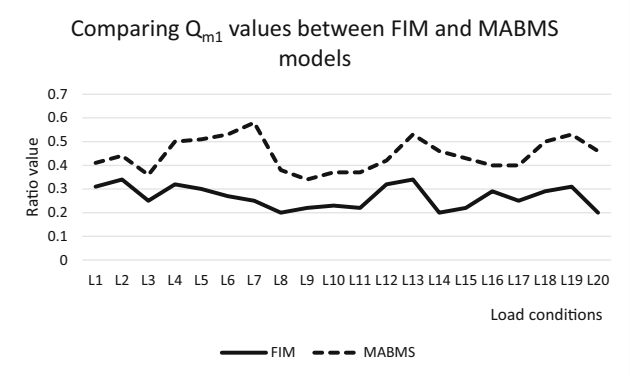

Fig. 55 Comparison of time ratio between FIM and MABMS models

In the case of autonomy, agent-based monitoring model is in a high zone, because mobile agents are goal driven and therefore collects status information autonomously. In the case of reliability, the agent-based monitoring system is in a high zone, because if a fault occurs then the mobile agent will automatically inform the monitoring node to take necessary action. FIM model is in a high zone because it uses mobile agents as well as fuzzy logic based decision module which increases the reliability of the overall system. In the case of mobility, agent-based monitoring model and FIM model are in the high zone, because both the models use mobile agents. In case of network latency, agent-based monitoring model and FIM model are in the low zone, because in both the models mobile agents carry the execution code to the target node and only results are sent back to monitoring nodes. Thus, the overall data communication is reduced resulting in a low network latency. In the case of accuracy, agent-based monitoring model is in the medium zone, because the mobile agent decision module cannot yield fine grain information from the collected data. The FIM model is in the high zone by employing the fuzzy logic based decision module to collect fine-grained information that is accurate and precise in nature. In the case of efficiency, agent-based monitoring model is in the middle zone, because the mobile agents in this model only collect coarse-grained information for decision-making. However, the FIM model efficiency is in the high zone, because in this model mobile agents are used for collecting status of available resources and fuzzy logic is used for decision making which enhances the efficiency of the overall system.

8.1.1 Quantitative analysis of MABMS and FIM models

In this section, we will quantitatively analyze our proposed models. The parameters for the comparative study is based on

Table 4 Comparison of total execution time, decision time, Decision output value and, computed ratio of FIM and MABMS models

Load			Total Execution Time		Decision Time		Decision output value		Computed ratio	
L	CPU %	RAM %	Agent-based monitoring model (*TE*)	Fuzzy integrated model (*TE*)	Agent-based monitoring model (*TD*)	Fuzzy integrated model (*TD*)	Agent-based monitoring model Q_{m2}	Fuzzy integrated model Q_{m2}	Agent-based monitoring model Q_{m1}	Fuzzy integrated model Q_{m1}
L1	0.04	0.8	1391	2661	577	851	0.032	0.2	0.41	0.31
L2	0.08	0.76	1284	2616	271	892	0.06	0.18	0.44	0.34
L3	0.12	0.72	1566	2604	564	654	0.08	0.17	0.36	0.25
L4	0.16	0.68	1355	2599	682	836	0.01	0.27	0.50	0.32
L5	0.20	0.64	1336	2595	690	794	0.12	0.26	0.51	0.30
L6	0.24	0.60	1206	2619	640	726	0.14	0.24	0.53	0.27
L7	0.28	0.56	1125	2599	656	674	0.15	0.26	0.58	0.25
L8	0.32	0.52	1497	2598	576	530	0.16	0.30	0.38	0.20
L9	0.36	0.48	1497	2614	510	590	0.172	0.37	0.34	0.22
L10	0.40	0.44	1523	2613	577	601	0.176	0.40	0.37	0.23
L11	0.44	0.40	1391	2605	528	589	0.176	0.43	0.37	0.22
L12	0.48	0.36	1209	2435	508	801	0.172	0.49	0.42	0.32
L13	0.52	0.32	1501	2439	800	830	0.16	0.53	0.53	0.34
L14	0.56	0.28	1140	2694	533	563	0.15	0.56	0.46	0.20
L15	0.60	0.24	1430	2504	615	558	0.14	0.59	0.43	0.22
L16	0.64	0.20	1391	2540	557	753	0.12	0.65	0.40	0.29
L17	0.68	0.16	1349	2617	552	671	0.1	0.69	0.40	0.25
L18	0.72	0.12	1240	2498	626	730	0.8	0.73	0.50	0.29
L19	0.76	0.08	1125	2531	605	789	0.6	0.77	0.53	0.31
L20	0.8	0.04	1374	2620	633	546	0.32	0.84	0.46	0.20

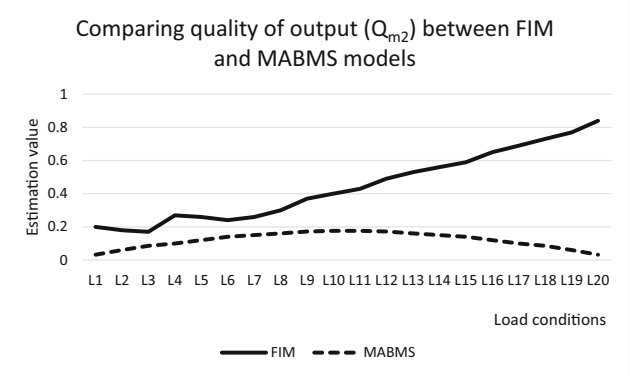

Fig. 56 Comparison of quality of output between FIM and MABMS models

total execution time, decision time, the ratio of decision/execution time, and quality of output as illustrated in Figs. 53, 54, 55 and, 56 respectively. Analysis for estimating total time execution for both the models are subject to have the same load as well as the same time intervals. The aim of this experiment is to analyze the time complexity of our models. It is observed that measuring time complexity the average execution time of the FIM model is slightly higher than the MABMS model as illustrated in Fig. 53. In the FIM model, decision-making is based on current status information for decision making. The agents collect status information and the fuzzy decision module use this data for decision-making. This functional behavior leads to high intercommunication resulting in high time complexity between connected nodes. The MABMS model uses only an agent for both collecting status information as well as decision making resulting in a decrease in time complexity. We have used predefined load status data for analyzing the decision time of our proposed models. In this experiment, the agent collects and submit predefined status information to decision module for

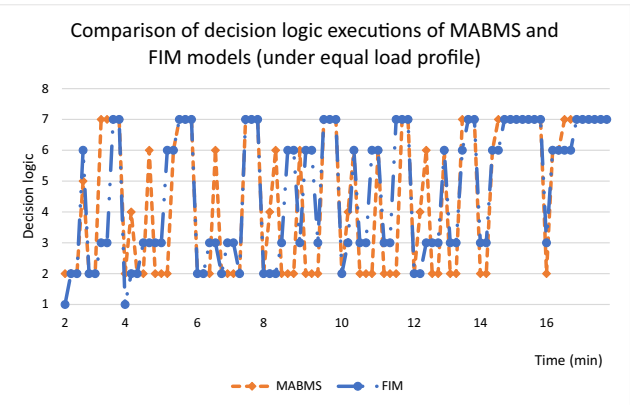

Fig. 57 Comparison of decision logic execution of FIM and MABMS models

decision-making. It is observed that the decision time of MABMS is significantly low as compared to the FIM model, as illustrated in Fig. 54. The MABMS model computes coarse-grained information consuming less amount of time in decision making, while FIM model computes fine grain information for decision and thus consuming more time in decision-making. Computing quality of output, we have considered three parameters characterizing as, decision time (T_D), total execution time (T_E) and, decision output value. Moreover, to perform this experiment, we have considered two quality metric Q_{m1} and Q_{m2}. The quality metric Q_{m1} is the computed ratio (T_D / T_E) and Q_{m2} is the computed estimation of real numbers. In the case of MABMS model, Q_{m2} is a probabilistic normed value and in the FIM model, Q_{m2} is the defuzzified output value. This quality metric estimation is done by a variety of different load conditions ranging from L1 to L20. L1 starts from lowest CPU and highest RAM (CPU: 0.04%, RAM: 0.8%) to the transition of the load distribution, while L20 is the opposite situation i.e. highest CPU and lowest RAM (CPU: 0.8%, RAM: 0.04%) as illustrated in Table 4. It is observed that the ratio of quality matrix Q_{m1} MABMS model is higher than the FIM model as illustrated in Fig. 55. In the FIM model, the time ratio is a bit high due to complex decision-making logic however, the output value is comparatively low under the same inputs. Complex decision logic evaluates a user-defined condition and then runs a sequence of test steps subject to return values of the condition. The condition used in the complex decision logic compares two values and returns a single value based on this comparison. The complex decision logic can be created by nesting many levels of condition statements as per required logic. In the case of quality matrix Q_{m2}, the experiment is carried out to observe the accuracy of FIM and MABMS models. In this experiment, a set of predefined load status data is used to observe the decision-making behavior at different load levels (L) as illustrated in Table 4. The FIM model possesses a high accuracy of decision-making as compared to MABMS model as illustrated in Fig. 56. The high accuracy is due to usage of smooth function that collects fine grain information for decision-making. The non-smooth function has sharp boundaries restricting from collecting fine-grained information on a continuous basis. Therefore, these functions are not differentiable everywhere. Thus, by losing valuable information not only affects the performance but also decreases the accuracy of decision-making. To find the relation between FIM and MABMS model we compute the cross-correlation of quality output values. Cross-correlation is used to compare two different datasets to detect if there is any relation between them having the same maximum and minimum values. To identify the level of correlation between two data sets i.e. quality of output value of the FIM model as well as MABMS model, we use normalized cross-correlation. The normalized cross-correlation can be computed as,

Table 5 Comparative analysis of load monitoring models

Parameters Agent-Based Models	Autonomousity	Reliability	Mobility	Network latency	Flexibility	Heterogeneity
ABAMS	H	L	H	H	H	H
MABNMMS	H	H	H	M	L	H
MonALISA	H	H	H	L	M	H
MABSRMS	H	M	H	M	L	M
DMSBA	M	H	H	M	H	H
MABLM	H	M	H	M	L	H
LISA	H	H	H	L	H	H
MABMS	H	H	H	L	M	H

H High, *M* Medium, *L* Low, agent-based adaptive monitoring system (ABAMS), Mobile agent-based network monitoring and management system (MABNMMS), Monitoring Agents in A Large Integrated Services Architecture (MonALISA), Mobile Agent-Based Server Resource Monitoring System (MABSRMS), Distributed Management System based on java agents (DMSBA), Mobile Agent-Based Load Monitoring (MABLM), Localhost Information Service Agent (LISA), Mobile Agent-Based Monitoring System (MABMS)

$$norm\_corr(a,b) = \frac{\sum_{n=L1}^{L20} a[n] \times b[n]}{\sqrt{\sum_{n=L1}^{L20} a[n]^2 \times \sum_{n=L1}^{L20} b[n]^2}}$$

Where (a) denotes FIM model quality output value and (b) denotes MABMS model quality output value where the sample length (n) is 20. By using the above formula, we compute the normalized cross-correlation for the FIM and MABMS models. Which shows that the FIM and MABMS models correlate, with a value of 0.155. The correlation between two variable is said to be perfectly correlated if it is equal to 1, if they are perfectly anticorrelated then −1 and, if they are completely uncorrelated then the value is 0. The correlation of FIM and MABMS is 0.155, which indicates that two models are correlated but are not perfectly correlated.

8.1.2 Comparison of decision logic executions

In this section, the rule firing density is evaluated to determine the performance of decision logic of MABMS and FIM algorithm. In this experiment, we have employed CPU stress and Heavy load benchmark to compute the combined load stress profile of CPU and RAM under the same load profiles. The total experimental execution time is sixteen minutes and after every two minutes, we change the load stress profile to evaluate the performance. The x-axis represents the time in a millisecond and y-axis represent the decision logic as shown in Fig. 57. For the comparison we have selected seven rules namely as, Any_load, light_cpu_ram_load, light_ram_load, light_cpu_load, load_cpu, load_ram, and stop_sending_load. As shown in the Figure, 70 to 75% the rule firing density of the decision logic remain the same. However, due to collecting fine-grained information the rule firing density of FIM algorithm is comparatively low. On the other hand, MABMS algorithm collects coarse-grained information, which results in high rule firing density as illustrated in Fig. 57.

8.1.3 Qualitative analysis with other models

In our qualitative analysis, we have selected six parameters for qualitative analysis namely, autonomous, reliability, mobility,

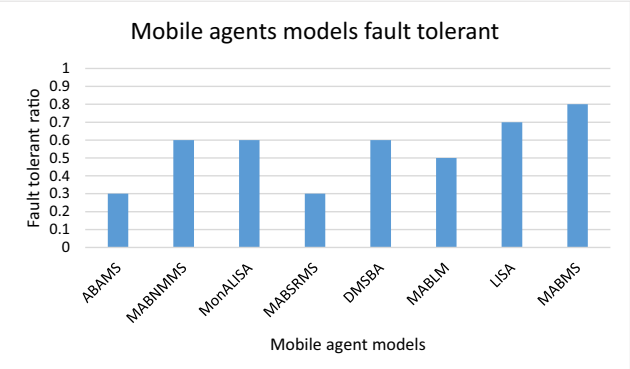

Fig. 58 Fault tolerance of mobile agent models

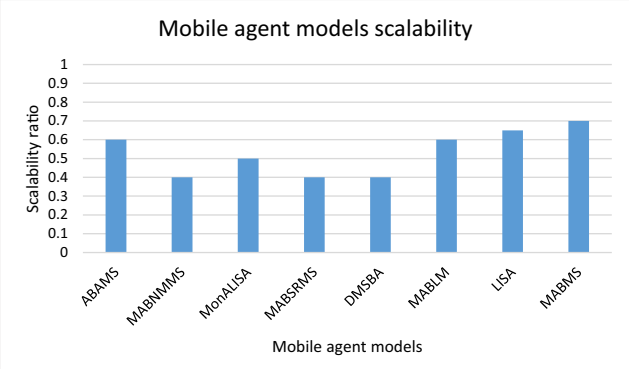

Fig. 59 Scalability distribution of mobile agent models

network latency, flexibility, and heterogeneity as illustrated in Table 5. Moreover, we have selected seven models for comparing with our proposed MABMS model. This set of models consisting of, *Agent-Based Adaptive Monitoring System* [38], *Java Based Agent Management System* [39], *Localhost Information Service Agent* (LISA) [29], *Web Server Load Monitoring System using a Mobile Agent* [28], *A Method of Network Monitoring by Mobile Agents* [11], *A Mobile Agent-Based System for Server Resource Monitoring* [14], *Mobile Agent-Based Load Monitoring* [40], and *Monitoring Agents in A Large Integrated Services Architecture* (MonALISA) [41]. These models have employed mobile agents for data collection, fault tolerance, efficiency, autonomous behavior, reliability and, scalability. It is observed that our model is in high zones except for when it is in a low (L) zone concerning network latency and in medium (M) zone concerning flexibility. The network latency is in the low zone because of agent-based controlled communication with the monitoring node and client nodes. Flexibility is in medium (M) zone because in our model changes in runtime environment are not allowed to specify additional functionalities.

8.1.4 Quantitative analysis with other models

In this section, we will quantitatively analyze different agent-based monitoring models. The comparative studies are performed based on fault tolerance, scalability, and response time as illustrated in Figs. 58, 59 and, 60, respectively. The values of performance metric are determined with approximation by analysis. As the fault tolerance, scalability, and response time is not quantifiable by using any exact analytical model, hence the method of empiricism (empirical quantitative analysis) is used. In our empirical estimation, the discretely quantized mapping is used in a unit interval. The metric values are set on a scale between 0 and 1, where 0 represents the lowest value and, 1 represents the highest value. We have divided the interval into three zones such as the first zone is from 0 to 0.3 (low zone), the second zone is from 0.3 to 0.6 (medium zone) and, the last zone is from 0.6 to 1 (high zone). In Fig. 58,

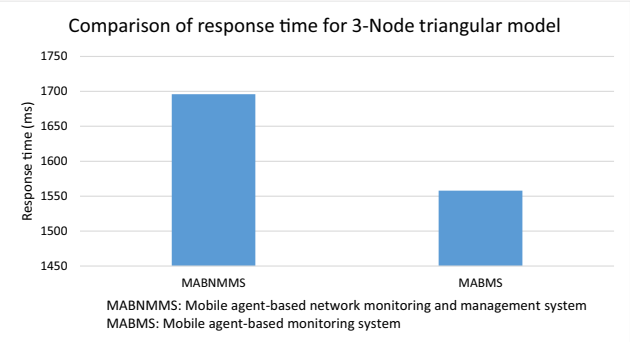

Fig. 60 Comparison of response time for 3-Node triangular model

ABAMS model is in a low zone in terms of fault tolerance because there is no such mechanism exists that could be used to recover from failures (i.e. low fault tolerant). MABNMMS monitoring model is in the medium zone because in this model the fault management uses mobile agents to monitor the network to identify faults and take necessary recovery action [11]. MonALISA framework is in the medium zone because this framework assigns an independent thread to each task so that if some tasks fail due to an error the other tasks should not be affected. MABSRMS monitoring framework is in a low zone because it uses more than one server to avoid failures, however there is no other efficient mechanism to make this framework fault tolerant. DMSBA is in the medium zone because this model uses broker agents that detect failures or congestion automatically and change the granularity at which the data is collected. MABLM is in the medium zone because this model uses a master agent to keep track of slave agents in case of failure. LISA framework is in the high zone because the core system is used to monitor the deployed module. The function of the core system is to restart the module in case of failures or malfunctions due to some running conditions or programming faults. In Fig. 59, the scalability of ABAMS framework is in the medium zone because it can monitor more than one multicast groups and the administrator can add or remove multicast group without disturbing the system performance [38]. MABNMMS monitoring model is in a low zone because this framework is tested on a fixed number of nodes, however, there is no data that shows that how it will perform in large-scale distributed systems. MonALISA is in the medium zone because this model is based on reusable threads upon completion of assigned tasks and these threads can be assigned to other nodes. MABSRMS monitoring framework is in the high zone because this model is designed for large-scale distributed systems. LISA framework is in the high zone because the agents automatically detect the underlying architecture and load binary modules necessary to perform monitoring services.

In Fig. 60, the comparison of response time for the 3-node model is illustrated considering MABNMMS and MABMS. As illustrated in Fig. 60, the response time of MABMS is less than MABNMMS, which shows that MABMS performance is better than MABNMMS. The results illustrate that the response time is dependent on the complexity of script as well as the number of available nodes in the network. Therefore, the response time increases with the increase in a number of nodes in the network and vice versa. While MABMS uses Java-based agents that are platform independent and lightweight agents. The Java-based agent migrates to various nodes along with its state and code. The mobile agent collects status information and, sends it to the monitoring node. Next, it migrates to the next available node in the network. Thus, this mechanism reduces the response time of nodes. On the other hand, MABNMMS executes on all available nodes, collects status

information, and sends the results to monitoring node increasing response time.

9 Conclusion

In large-scale distributed systems, manually keeping track of the system load variations is a challenging task in terms of management and accuracy of decision-making. In this paper, we have developed and implemented two different resource estimation and monitoring models. In the mobile agent-based model, the pure agent-based approach is used for load estimation by employing a joint probability based estimation of resources and a norm function, which is computationally inexpensive. This model can be used in a time-critical system where low time complexity is required. Moreover, the mobile agent is lightweight, goal oriented and autonomous software entity, which consumes fewer network resources. In addition, our proposed agent-based model is characterized by having decreased time complexity as well as improved system performance. The second model is an integrated load balancing and monitoring model employing type-1 fuzzy logic. The fuzzy integrated model is characterized by smooth and composite fuzzy membership function to model fine-grained load information. Moreover, the fuzzy integrated model can be used in a system where the precision and accuracy of decision making is required. Comparative studies of performances of the two models are carried out and results are presented in this paper. The comparative analysis illustrates that the fuzzy integrated model is preferable due to better accuracy of decision making and to achieve better execution time. We have measured the cross-correlation of performances between two models to find out the variations of quality of output values.

The future work includes the implementation of type-2 fuzzy logic mechanism in our fuzzy integrated model. The type-2 fuzzy logic approach will handle more precisely and accurately the rule of uncertainties in determining load monitoring and load balancing. However, implementing this approach will be challenging by keeping reduced computational time complexity along with reduced computational cost. The high computational cost of the iterative algorithm means that it is more expensive to deploy type-2 fuzzy logic based approach. In our fuzzy integrated model, we have to implement a fuzzy decision module in the monitoring node. In the future work, we will implement fuzzy decision module embedded in software agents for decision making on the destination node. The advantage of this approach will be to reduce the computation at the monitoring node. Moreover, in this approach, we will introduce a backup mechanism to overcome the problem of single point of failure.

References

1. Xu F, Liu F, Liu L, Jin H, Li B (2014) iAware: making live migration of virtual machines interference-aware in the cloud. IEEE Trans Comput 63:3012–3025
2. Rajani S., and Garg N. A clustered approach for load balancing in distributed systems, international journal of Mobile Computing & Application, volume: 2, SSRG-IJMCA, 2015, ISSN: 2393-9141
3. Wörn H, Längle T, Albert M, Kazi A, Brighenti A, Seijo SR, Senior C, Bobi MAS, Collado JV (2004) Diamond: distributed multi-agent architecture for monitoring and diagnosis. Prod Plan Control 15(2): 189–200
4. Tomarchio, O., Vita, L. and Puliafito, A., Active monitoring in grid environments using mobile agent technology, In *Active Middleware Services*, Springer, Boston, MA, 2000, pp. 57–66
5. Haverkamp DS, Gauch S (1998) Intelligent information agents: review and challenges for distributed information sources. J Assoc Inf Sci Technol 49(4):304–311
6. Alakeel AM (2016) Application of fuzzy logic in load balancing of homogenous distributed systems. Int J Comput Sci Secur IJCSS 10(3):95–101
7. Alipour MM, Derakhshi MRF (2016) Two level fuzzy approach for dynamic load balancing in the cloud computing. J Electron Syst 6(1):17–31
8. Ahn, H.C., Youn, H.Y., Jeon, K.Y. and Lee, K.S. Dynamic load balancing for large-scale distributed system with intelligent fuzzy controller, In *Information Reuse and Integration, IEEE International Conference on,* IEEE, 2007, pp. 576–581
9. Das S (2013) Mobile agents in distributed computing: network exploration. Bulletin of EATCS 1:109
10. Papavassiliou S, Puliafito A, Tomarchio O, Ye J (2002) Mobile agent-based approach for efficient network management and resource allocation: framework and applications. IEEE J Sel Areas Commun 20(4):858–872
11. Manvi SS, Venkataram P A method of network monitoring by mobile agents. Computing *2*(3):4–5
12. Adacal M, Bener AB (2006) Mobile web services: a new agent-based framework. IEEE Internet Comput 10(3):58–65
13. Du TC, Li EY, Chang AP (2003) Mobile agents in distributed network management. Commun ACM 46(7):127–132
14. Aridor, Yariv and Danny B. L. Agent design patterns: elements of agent application design, In Proceedings of the second international conference on Autonomous agents, ACM, 1998, pp. 108–115
15. Mostafa, Salama A., Mohd S. A., Muthukkaruppan A., Azhana A., and Saraswathy S. G. A dynamically adjustable autonomic agent framework, In Advances in Information Systems and Technologies, Springer, 2013, pp 631–642
16. Ku H, Luderer GW, Subbiah B (1997) An intelligent mobile agent framework for distributed network management. In Global telecommunications conference, GLOBECOM'97. IEEE 1:160–164
17. Corradi A, Cremonini M, Montanari R, Stefanelli C (1999) Mobile agents integrity for electronic commerce applications. Inf Syst 24(6, Elsevier):519–533
18. Ahn J (2010) Fault-tolerant Mobile agent-based monitoring mechanism for highly dynamic distributed networks. IJCSI Int J Comput Sci Issues 7(3):1–7
19. Park HJ, Jyung KJ, Kim SS (2004) Mobile agent-based load monitoring system for the safety web server environment. In: In international conference on computational science. Springer, pp 274–280
20. Wang X, Wang H, Wang Y (2010) A unified monitoring framework for distributed environment. Intell Inf Manag 2(07):398–405
21. Massie ML, Chun BN, Culler DE (2004) The ganglia distributed monitoring system: design, implementation, and experience. Parallel Comput 30(7):817–840

22. Vidhate SL, Kharat MU (2014) Resource aware monitoring in distributed system using Tabu search algorithm. Int J Comput Appl 96(23):22–25
23. Tomarchio, O. and Vita, L. On the use of mobile code technology for monitoring grid system, In *Cluster Computing and the Grid, Proceedings First IEEE/ACM International Symposium on,* IEEE, 2001, pp. 450–455
24. Iosup, A., Ţăpuş, N. and Vialle, S. A monitoring architecture for control grids, In European Grid Conference, Springer, 2005, pp. 922–931
25. Mace, J., Roelke, R. and Fonseca, R. Pivot tracing: dynamic causal monitoring for distributed systems, In Proceedings of the 25th symposium on operating systems principles, ACM, 2015, pp. 378–393
26. Gunter, D., Tierney, B., Jackson, K., Lee, J. and Stoufer, M. Dynamic monitoring of high-performance distributed applications, In High performance distributed computing, 11th IEEE international symposium, IEEE, 2002, pp. 163–170
27. Hoke E, Sun J, Faloutsos CI (2006) Intelligent system monitoring on large clusters. In: Proceedings of the 32nd international conference on very large data bases, VLDB endowment, ACM, pp 1239–1242
28. Tie Z (2013) A Mobile agent-based system for server resource monitoring. Cybernetics and Information Technologies 13(4): 104–117
29. Dobre, C., Voicu, R., Muraru, A., and Legrand, I.C. A distributed agent based system to control and coordinate large scale data transfers, 2011, arXiv preprint arXiv:1106.5171, 2011
30. Seenuvasan P, Kannan A, Varalakshmi P (2017) Agent-based resource management in a cloud environment. Appl Math 11(3):777–788
31. Helmy T, Al-Jamimi H, Ahmed B, Loqman H (2012) Fuzzy logic-based scheme for load balancing in grid services. J Softw Eng Appl 5:149–157
32. Floyd MW, Esfandiari B (2018) Supplemental observation acquisition for learning by observation agents. Appl Intell, Springer 48:1–17. https://doi.org/10.1007/s10489-018-1191-5
33. Zhong W, Zhuang Y, Sun J, Gu J (2018) A load prediction model for cloud computing using PSO-based weighted wavelet support vector machine. Appl Intell, Springer 48:1–12. https://doi.org/10.1007/s10489-018-1194-2
34. Bagchi S (2016) Probabilistic and fuzzy process classifiers for operating systems scheduler. Fundamenta Informaticae 145(4):405–427
35. Rahmat RS, Lafuerza Guillén B (2009) Probabilistic norms and statistical convergence of random variables, surveys in mathematics and its applications, vol 4, pp 65–76
36. Nine MSZ, Azad MAK, Abdullah S, Rahman RM (2013) Fuzzy logic based dynamic load balancing in virtualized data centers. In: Fuzzy systems (FUZZ), 2013 IEEE international conference. IEEE, pp 1–7
37. Velde, V. and Rama, B. An advanced algorithm for load balancing in cloud computing using fuzzy technique, In *Intelligent Computing and Control Systems (ICICCS), International Conference, IEEE,* 2017, pp. 1042–1047
38. Kwon, S. and Choi, J. An agent-based adaptive monitoring system, In Pacific rim international workshop on multi-agents, Springer, Berlin, Heidelberg, 2006, pp. 672–677
39. Brooks, C., Tierney, B. and Johnston, W. JAVA agents for distributed system management, LBNL Report, 1997
40. Kim ST, Park HJ, Kim YC (2001) The load monitoring of web server using mobile agent, in *Info-tech and Info-net, 2001, Proceedings. ICII 2001-Beijing*. International Conferences on, IEEE 5:89–94
41. Legrand I, Newman H, Voicu R, Cirstoiu C, Grigoras C, Dobre C, Muraru A, Costan A, Dediu M, Stratan C (2009) MonALISA: an agent based, dynamic service system to monitor, control and optimize distributed systems. Comput Phys Commun 180(12):2472–2498

Moazam Ali obtained his BCS in Computer Science in year 2007 from the Islamia College, University of Peshawar and, MS-IT in Computer Networks in year 2012 from the Institute of Management Sciences, Peshawar. Currently, he is pursuing his PhD in Distributed Systems in the Department of Aerospace and Software Engineering (Informatics), Gyeongsang National University, Jinju, South Korea.

Susmit Bagchi has received B.Sc. (Honours) from Calcutta University in 1993, B. E. in Electronics Engineering in 1997 from Nagpur University, M.E. in Electronics and Telecommunication Engineering in 1999 from Bengal Engineering and Science University (presently IIEST). He has obtained Ph.D. (Engineering) in Information Technology in 2008 from Bengal Engineering and Science University. Currently, he is Associate Professor in Department of Aerospace and Software Engineering (Informatics), Gyeongsang National University, Jinju, South Korea. His research interests are in Distributed Computing Systems and Analysis.