Accepted Manuscript

Probabilistic normed load monitoring in large scale distributed systems using mobile agents

Moazam Ali, Susmit Bagchi

PII: S0167-739X(18)30984-1
DOI: https://doi.org/10.1016/j.future.2019.01.053
Reference: FUTURE 4746

To appear in: *Future Generation Computer Systems*

Received date : 25 April 2018
Revised date : 18 December 2018
Accepted date : 27 January 2019

Please cite this article as: M. Ali and S. Bagchi, Probabilistic normed load monitoring in large scale distributed systems using mobile agents, *Future Generation Computer Systems* (2019), https://doi.org/10.1016/j.future.2019.01.053

Probabilistic Normed Load Monitoring in Large Scale Distributed Systems using Mobile Agents

Moazam Ali, Susmit Bagchi
Department of Aerospace and Software Engineering (Informatics), Gyeongsang National University, Jinju, South Korea 660701
Corresponding author: Susmit Bagchi, profsbagchi@gmail.com

Abstract

Due to rapid advancements and developments in computing platforms, there is a tremendous growth in heterogeneous distributed systems involving mobile devices. In distributed systems, achieving better performance and efficient utilization of shared resources are dependent on appropriate load monitoring mechanisms. In large-scale distributed systems, performing load monitoring is a challenging task and, it effects in increasing response time degrading system performance. In this paper, we have developed and implemented mobile agent-based load monitoring system intended to large-scale distributed systems. Mobile agent based design is an attractive approach for load monitoring in large-scale distributed systems, because mobile agents are autonomous, goal-driven, reactive to environmental changes and, lightweight piece of program. In this paper, a detailed software architectural design for mobile agent based load monitoring system is presented. The design is based on a probabilistic normed estimation model and corresponding monitoring algorithms. The experimental evaluations and scalability analysis presented in this paper illustrate the behavior of agents and performance of the system under different load conditions. Moreover, a detailed qualitative as well as quantitative analysis of various mobile agent models are presented.

Keywords: Distributed Systems, Mobile Agents, Load Monitoring, Resource Utilization, Cloud Computing.

1. INTRODUCTION

A distributed system is the combination of independent nodes operating in a network and appears to the users as a single coherent system [6]. Distributed systems mean distributed processing in a shared environment to minimize computing time and to increase the overall performance. However, achieving better performance and reduction of computing time are subject to proper load balancing and load monitoring mechanisms [7, 31]. It means that, the overall processing load should be equally distributed among all the nodes with the proper load monitoring mechanism to avoid underload or overload conditions. In recent time, researchers have proposed that strict load balancing in a distributed system may not offer improved performance [34]. However, in general, it is observed that load balancing is an important factor in heterogeneous large-scale distributed systems to avoid overloading/underloading skews at nodes [32, 33]. In distributed systems, load monitoring mechanisms are used to monitor available computing nodes in a system [25]. Without load monitoring, it is difficult to employ load balancing approaches to distributed systems. In large-scale distributed systems, load monitoring by a system administrator is a very difficult task [7, 26]. Moreover, load monitor creates frequent intercommunication in distributed systems while collecting node information repeatedly [8].

Furthermore, checking the current status of the large set of available nodes in a distributed systems is difficult in the presence of network latency, because getting the current status of all the available nodes incurs immense internetwork communications increasing response time and affecting system performance. Therefore, a mechanism is required to overcome these problems in determining the current status of the available nodes. Mobile agents are emerging technologies which has the ability to autonomously manage, design, implement and maintain distributed systems [3, 28]. Mobile agents are autonomous software entities having the ability to migrate through the network from node to node [1, 24]. The basic characteristics of mobile agents are autonomy in behavior, social interaction, reactive to its environmental changes and goal driven execution [2]. A mobile agent created in one node can transport its "code" and "state" to another node in the network, where it continues its execution [5, 29]. The function of "code" is to start execution and the "state" determines the actions of a mobile agent in the destination node. Thus, mobile agent based load monitoring is an attractive approach in large-scale distributed systems.

1.1 MOTIVATION

Load monitoring mechanisms are designed to monitor distributed processing load to improve system performance and to enhance resource utilization in distributed systems [27]. In order to maximize the efficiency of distributed systems and for better utilization of the resources load monitoring mechanisms should be able to compute the current status of nodes. Researchers have proposed different approaches to load monitoring in distributed systems [1, 4]. A Grid Monitoring Architecture (GMA) is designed for grid computing by Global Grid Forum Performance Working Group [9]. The key feature of GMA is to provide a low-level specification that supports required functionality and, enables interoperability. However, in general, the GMA model is computationally expensive [9]. In GMA, intercommunication between producer and consumer for a large set of tasks would enhance computational complexity to a large extent. The space complexity of the GMA model is high because of the information storage including a set of events. The Ganglia is a scalable distributed monitoring system design for clusters and grids to achieve high-performance computing systems [10]. Ganglia is based on the hierarchical design to form a federation of clusters and it depends on a multicast based listen or announce protocol to monitor the state of clusters as well as cluster nodes. In ganglia, there is no mechanism to store information of producers and consumers. Tabu search algorithm is another neighborhood based search method which is used in distributed systems for monitoring applications [11]. However, a constant collection of detailed performance attribute values of a large number of nodes is difficult. On the contrary, we argue that the mobile agent-based load monitoring approaches have several advantages such as [11, 30], (a) Reduced network load, (b) Network delay resolve, (c) Dynamic adaptation (d) Fault tolerance and, (e) Goal-driven behavior.

In this paper, we present the design, implementation, and evaluation of mobile agents based load monitoring in a distributed system environment. The process load estimation is computed by employing a joint probability model and norm function, which is computationally inexpensive. The detailed scalability analysis is presented by using experimental evaluations as well as by following regression analytical model. The distinctive features of the proposed approach are,

- Autonomous load monitoring by migrating mobile agents based on the probabilistic norm.
- Reduction of waiting time of a node as well as the network load in order to increase the overall system performance.
- Updating real-time load information to monitoring node for decision making based on time intervals.
- Adaptive decision making by mobile agents depending on varying status of a node.

Rest of the paper is organized as follows. Section 2 presents related work. Section 3 illustrates software architectural design and estimation model. Section 4 presents monitoring algorithms. Section 5 describes the implementation environment. Section 6 represents experimental evaluations. Section 7 presents a comparative analysis of the proposed algorithm to other contemporary designs. Lastly, section 8 concludes the paper.

2. Related Work

There are various approaches available for load monitoring in distributed systems. In general, load monitoring in large-scale distributed systems requires a considerable amount of computational resources [7]. The researchers have proposed model-based methods and algorithms for minimization of time and energy of computations for profiling data related to performance and energy of servers having multicore processors [35]. However, the algorithmic model considers specific server configuration for evaluation and, does not cover heterogeneous nodes including clients. Iosup, A et al. have proposed a monitoring architecture for control of grids, which is known as Toytle [16]. The core issues addressed by Toytle are grid-awareness, scalability and communication standards. Toytle architecture inherits Global Grid Forum Grid Monitoring Architecture (GGFGMA) guidelines [16]. 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 [17]. 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. The Pivot tracing combines dynamic instrumentation with a relational operator e.g. happened before join which gives users run-time ability to define arbitrary metrics. The Pivot tracing approach is able to filter, select and group based on events meaningful at other parts of the systems. Dan Ganter et al. have proposed a very lightweight instrumentation system for dynamic monitoring of high performance distributed applications [18]. 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. Their proposed approach is used to deal with performance problems such as, low throughput and high latency by

determining a detailed end to end instrumentation of all components including applications, operating systems, hosts and networks. InteMon is designed for monitoring and data mining in large clusters [19]. 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 [19]. Mobile agents are an efficient approach, which migrates from node to node in a connected network for gathering 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 [12]. In MABSRMS, mobile agents call a low-level function to monitor server resources effectively and efficiently. The monitoring algorithm can be deployed on some nodes or all monitoring nodes quickly, easily and silently. 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 [13]. The hierarchical model gives the advantage to 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 [13]. Localhost Information Service Agent (LISA) is a lightweight dynamic service capable of providing application monitoring, configuring system parameter and optimizing distributed applications [14]. 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. This approach is capable to continuously monitor itself and detect any potential problems and selects a possible solution to overcome the problem. Precedence Based Monitoring (PBM) algorithm is proposed for load monitoring in cloud architecture to manage resources based on time, event and precedence [15]. 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.

3. Software Architecture Design

3.1. Monitoring Agent Model

The architectural model of the designed Mobile agent monitoring system is illustrated in Figure 1. The proposed architecture has three main modules such as, (a) Agent Monitoring Module (AMM), (b) Agent Decision Module (ADM) and, (c) Agent Migration Module (AMMO). The agent monitoring module is used to collect the current status of the available resources in distributed systems. In addition, the agent is programmed to wait for a randomly varying time intervals to generate different samples of the currently available resources of a node. The purpose of collecting different samples is to compare the status of resources at different time

intervals with randomness. The ADM computes these samples of data for making decisions. The decision can be at multiple levels. For example, to re-compute the resources if the condition for threshold doesn't match, otherwise forwarding it to AMMO. The AMMO is used to migrate the agent to appropriate nodes in distributed systems if a migration decision is made autonomously. The background processes or daemon processes are those programs that are periodic or non-terminable. Most of these processes are part of the operating systems. The background processes have significance in managing system performance. System resources are classified into different types and functions such as, CPU, video card, hard drive and, memory. In this paper, we are referring RAM and CPU as the main available system resources in a node for computational purposes.

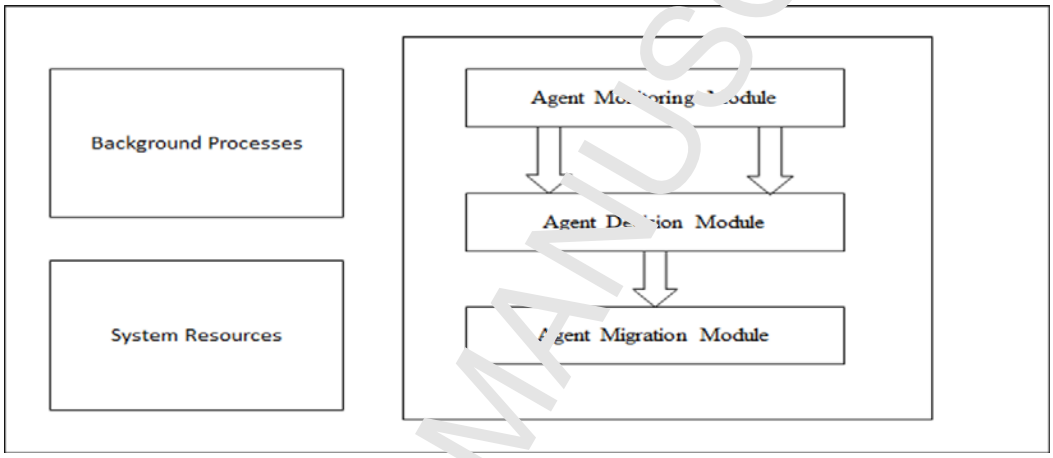

Figure 1: Schematic diagram of Mobile Agent Model

3.2. Architectural Design

The internals of architectural design of our proposed monitoring system based on mobile agents are shown in Figure 2. The agent monitoring module is consisting of sub-modules for systems data collection, for computing idle and used resources (RAM and CPU), randomization of sampling and, storage of sampling data for the decision-making process. The agent decision module is consisting of sub-modules for retrieving sampling data, categorization of sampling data by monitoring logic and decision making for agent migration. The agent migration module is consisting of sub-modules for collecting updated system loads, selection of the appropriate node based on specified criteria and migrating the agent to the next node for computing the current status of system resources.

- **Data collection module (DCM):** The function of the data collection module is to start the data extraction from the currently available resources. The mobile agent will collect the current status of resources i.e. RAM and CPU. In the first phase, the DCM module calculates the total size of RAM and CPU. In the second phase, the DCM module computes the total number of processes that are currently holding the resources. In the third phase, agents send this information to the resource compute module (RCM).
- **Resource compute module (RCM):** The resource compute module is used to compute the current status of the resources. The RCM module collects the status related

information from DCM in order to compute and convert it into a percentage value. Calculating percentage value helps to efficiently determine to know exactly the percentage usage of RAM and CPU is varying in random time intervals. Later on, this information is sent to the data sampling module.

- **Data sampling module (DSM):** The data sampling module will take the first sample and will send it to the storage buffer and then waits for a random amount of time. Next, the sampling module will continue and take the second sample from RCM and, this process will continue until the specified amount of sampling is done. 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 effects of rapidly transiting processes in a system, the randomization of waiting time is employed.
- **Storage buffer module (SBM):** The storage buffer 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. The SBM module will store the samples and then wait for the random amount of time to continue sample collection.

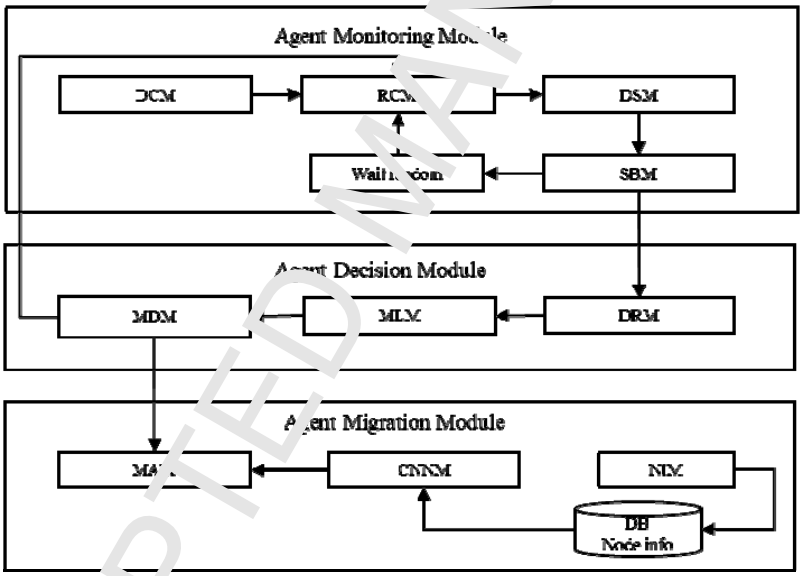

Figure 2. Mobile Agent Design Architecture

- **Data retrieval module (DRM):** The stored data is retrieved from the storage buffer by the data retrieval module. The stored data contains status information of available resources i.e. CPU status and RAM status, which indicate that how much CPU as well as RAM is loaded in a node. The DRM module will instantaneously retrieve the sampling data from SBM. The purpose of this module is to get the sampling data and to send it to the MLM module.
- **Monitoring logic module (MLM):** The monitoring logic module computes the joint probability variation of processing load in a node based on sampled data. The minimum joint probability value is 0 and the maximum joint probability is 1. The joint probability

is used to calculate two instantaneous events that are probably occurring at the same time. Thus, we are calculating joint random variations of CPU load and RAM load of a node. By taking the joint probability of these 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 monitoring logic is the key module of our mobile agent-based monitoring model, where monitoring logic will categorize the status of system load according to the embedded logic to achieve the desired goal.

- **Monitoring decision module (MDM):** The monitoring decision module is used to make a decision either to migrate the agent or to re-compute the status of resources. The monitoring decision module will make a decision based on the current status of CPU and RAM. For example, if the joint probability of the resource load of a node is greater than a specified threshold value, then the MDM will enforce the agent to enter into the migrating agent module (MAM). However, if the joint probability is less than the specified threshold value, then the MDM module will send the agent to execute the RCM module for re-computing the status information.
- **Node info module (NIM):** The function of the node info module is to check the current status of existing nodes in the distributed systems in order to determine the online or offline nodes. The NIM module is very important in our mobile agent monitoring architecture to ensure the reliability and to increase the performance of the system by regularly updating its database if a node is added or removed.
- **Computing next node module (CNNM).** The computing next node module is used to select the appropriate node based on the specified criteria. The CNNM module is a dynamic module because its criteria for selecting node can be easily changed according to the need of the system. The criteria for selecting an appropriate node can be categorized as, (a) select such a node which has high level of processing and computing resources, (b) select a node which has enough free available resources, (c) select nearest or adjacent node and, (d) a node which have fast and less congested network link. In our proposed model, we select the adjacent node to decrease the computation time.
- **Migrate agent module (MAM):** Migrate agent module is used to migrate the agent to the next node for computing the current status of the node. This module takes two inputs: one input from the migration decision module and a second input from computing next node module. The migration decision module will confirm the migration of agent, while the computing next node module will specify the node where the agent will migrate.

3.3. Load Estimation Model

In order to calculate the joint probability variations of CPU load and RAM load to get combined load status of the node n, the estimation model will compute the joint probability of CPU load $P_n(C)$ and RAM load $P_n(R)$ variations over time. Three samples are collected randomly within an estimation time window to reduce enlargement of memory consumption by statistical datasets. This set of samples is sufficient to formulate probabilistic normed estimation of dynamics of load variations. High frequency sampling in an interval is avoided because, repeated

samplings are done at multiple intervals and, past sampled datasets are temporarily kept in history for decision-making purposes. It effectively means that, more than three samples are taken into consideration for making decisions. Hence, the computation of the discrete joint probability of resource load in a time window at the node n is given below.

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

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,

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

It is easy to verify that Eqn. (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, $A_n = v_n + avg(d_n(a,b))$ 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. The Algorithm Design

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. The designing of the monitoring algorithm and decision algorithm are given below.

4.1. 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 Figure 3. 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. These values of cpu_load and ram_load are stored in an array for further processing. In the next step, the

algorithm accesses and retrieves the associated stored values in order to calculate the joint probability for cpu_load and ram_load of a node. The purpose of calculating joint probability is to determine the random variations of cpu_load and ram_load in order to generate a combined load status. The algorithm is designed to run and compute the joint probability for at least three different time instances to efficiently and effectively calculate the cpu_load and ram_load in random time intervals within an estimation time window.

```
Data types:
double mem, cpuperc, ram_per,cpu_per;
array cpu_load[], array ram_load[];
double v1, v2, v3,d12, d13, d23, vn, 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());
v1 = (cpu_load[0] . ram_load[0]);
v2 = (cpu_load[1] . ram_load[1]);
v3 = (cpu_load[2] . ram_load[2]);
d12 = | v1 − v2|;
d13 = | v1 − v3|;
d23 = | v2 − v3|;
vn = min(v1, v2, v3) ;
v = |min(vn) + ((d12+d13+d23)/3)|;
monitoring_decision(v);}
```

Figure 3. Pseudo-code representation of monitoring algorithm

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.

4.2. Decision Algorithms

The function of 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. The

distribution of zones is based on joint probability 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.

4.2.1. 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 pseudocode representation of a decision algorithm for zone_1 is presented in Figure 4.

```
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 v 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, ram,node_id >);
v_history = v_current;
migrate(Agent);}}}
```

Figure 4. Pseudocode representation of decision algorithm for zone_1

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 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 the 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.

4.2.2. 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 pseudocode representation of zone_2 of decision algorithm is represented in Figure 5.

```
//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_3) ∨ (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); }
}
```

Figure 5. Pseudocode representation of decision algorithm for zone_2

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

4.2.3. 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 representation of zone_3 of decision algorithm is presented in Figure 6.

```
//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);
}}
```

Figure 6. Pseudocode representation of decision algorithm for zone_3

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 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. Implementation Environment

5.1.1. Deployment Model

We have implemented our software architecture on testbed comprised of five nodes having completely connected topology. We have designated one node as a monitoring node and, remaining four nodes are targets executing mobile agents. The inherent properties of a mobile agent are autonomousity, goal orientation and, intelligent decision making [20]. The advantage of using mobile agents in distributed systems is to reduce the communication overhead by collecting the updated status of the available node resources. Our proposed deployment model for mobile agent monitoring system is illustrated in Figure 7.

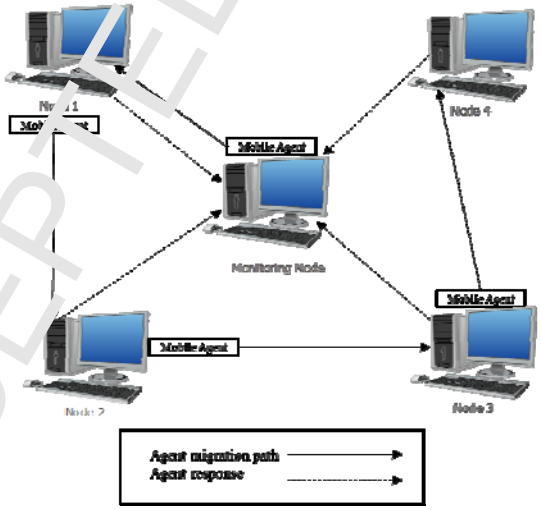

Figure 7. Deployment model of mobile agent monitoring systems

The mobile agent will first check the availability of a node where it will migrate next; if the node is alive then the agent will migrate. Otherwise, it will check for next available node. The node

information is stored a priori into the node information database. The database is updated when a node is added or removed from the system. The feature of our proposed agent monitoring system is to enable the mobile agent to efficiently collect the status information of a node and to send the information to the monitoring node. Another feature of proposed monitoring is the improved overall response time. In the traditional load monitoring approaches, the current load values are calculated for all the available nodes and later the aggregated information is sent as a response message to the monitoring node. On the contrary, our proposed model calculates the current status of nodes and immediately sends the information to the monitoring node. This mechanism will reduce the response time improving the overall system performance.

5.1.2. Implementation Architecture

Our proposed architecture is implemented in heterogeneous operating systems environment and Java programming language is used to deploy a mobile agent framework (i.e. Java Development Framework (JADE)). JADE is selected because it is a Java-based platform providing a simple and, efficient environment. Agents are implemented as containers in JADE and, are distributed among all the nodes in the network. The first container launched by the platform is known as "Main Container" and all other containers launched after the "Main Container" is named as "Container-1", "Container-2", "Container-3" and so on. The main container represents the bootstrap point. The containers are connected by the Internal Message Transport Protocol (IMTP). The main container components are *container table* (CT), *global agent descriptor table* (GADT), *local agent descriptor table* (LADT), *agent management system* (AMS) and, *directory facilitator* (DF), as illustrated in Figure 8. The container-1 components are *global agent descriptor table* (GADT) and, *local agent descriptor table* (LADT), as illustrated in Figure 8.

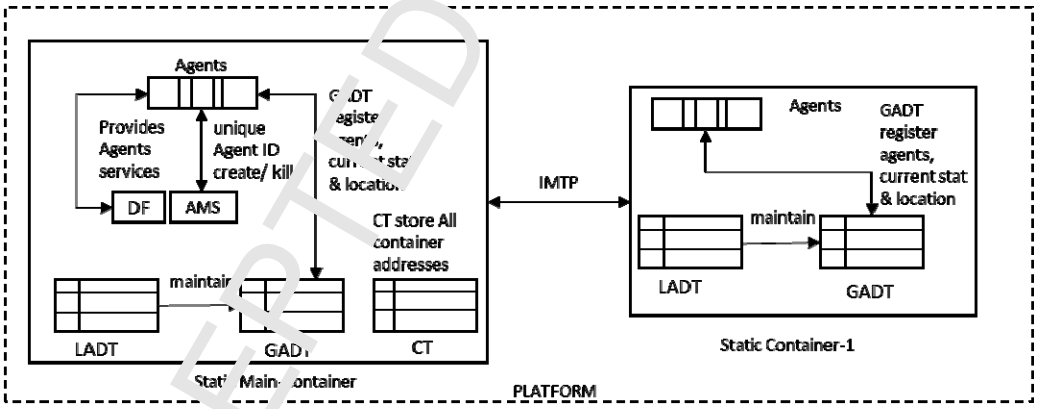

Figure 8. Schematic diagram of static main-container and static container-1

Moreover, the CT manages a registry of the object references and transport addresses of all available containers of the platform. GADT manages a registry of all available agents in the platform holding their current status and location. The LADT contains its local agent's address and also maintains global agent descriptor table. Agent management system monitors the entire platform and, it provides access to the white pages of the platform as well as manages the life cycle of the agent. Every agent must register with AMS in order to obtain valid AID. The

directory facilitator (DF) is the agent that implements the yellow pages service to any agent which wants to register its services or search for other available services.

The configuration of our mobile agent framework development and runtime environment specification is illustrated in Table 1. Our proposed mobile agent monitoring algorithm is developed using Java eclipse IDE on top of JADE agent platform for monitoring distributed system. 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 this software is to generate various categories of load on our target nodes to monitor the variations in algorithmic behavior under different load conditions. In terms of network connectivity, one of the nodes is wirelessly connected and the rest of the nodes are wired connected with the monitoring node. The wired connections operate at 100Mbps through switch and wireless network operates on 100Mbps at peak.

Table 1. Platform specifications of 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: 2TB, NIC: Realtek PCIe GBE Family Controller	Windows 10	
Network	Ethernet: 100Mbps LAN Wireless: 100Mbps WAP, Signal strength: 45% (average)		

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.

6. Experimental Evaluations

The monitoring algorithm computes the status of currently available node resources in a specified amount of time. In order to evaluate the behavior and performance of our implemented system, we have conducted several sets of experiments under different load conditions. We have employed four standard benchmarks namely, (1) Heavy load, (2) CPU stress (3) FFT-z and, (4) DGEMM. These benchmarks exert non-uniform load stress at nodes as illustrated in sampled profiles presented in Figure 9, Figure 10, Figure 31, Figure 33, Figure 35 and, Figure 37. The stress profile samples indicate that, CPU and RAM variations are uncorrelated and non-uniform in nature. We have evaluated the performance by using a set of parameters, which are zone dynamics, rule firing density, load dynamics, migration frequency under different load conditions, resource utilization and, decision logic execution. The rule firing density represents the total number of rules fired out of the entire set of rules in the time intervals. The rules are fired by the decision logic module of the mobile agent during system monitoring. The migration frequency represents the frequency of migration of agents between different nodes.

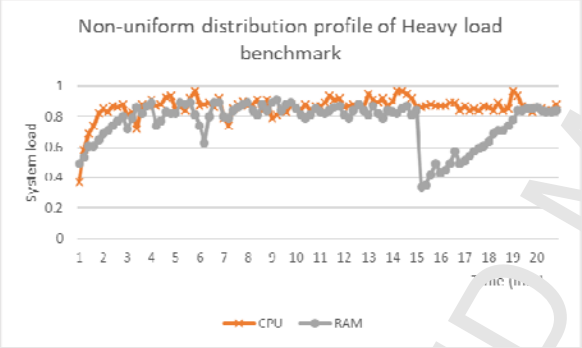

Figure 9. Non-uniform distribution profile of Heavy load benchmark

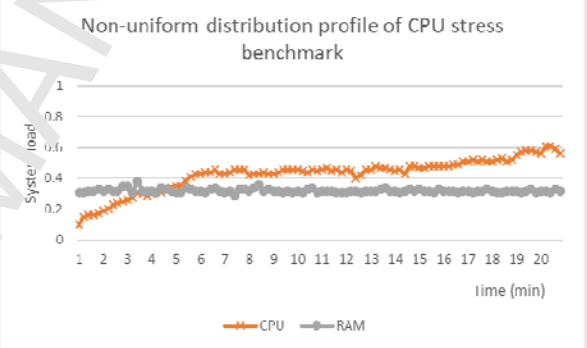

Figure 10. Non-uniform distribution profile of CPU stress benchmark

6.1. Set 1: Experimenting with CPU utilization in terms of Heavy Load and CPU Stress Benchmarks

In this case, the cpu_load is employed for estimating the behavior of zone dynamics, rule firing dynamics, the combined dynamics of CPU/RAM utilization, joint probability and, migration frequency of agents. The variation in zone dynamics with respect to increasing cpu_load is illustrated in Figure 11. The variations in rule firing density are illustrated in Figure 12. The variations in the combined dynamics of CPU/RAM and joint probability are illustrated in Figure 13. The variations in migration frequency are illustrated in Figure 14.

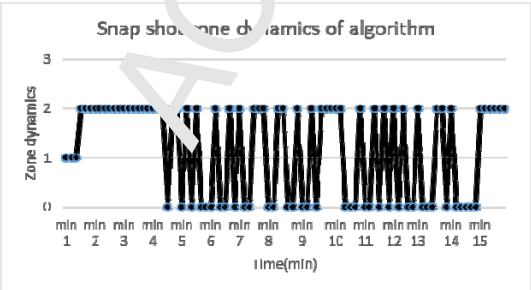

Figure 11. Zone dynamics of mobile agent monitoring system (ex-set-1)

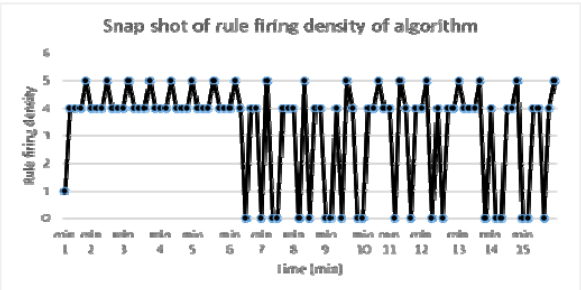

Figure 12. Rule firing density of mobile agent monitoring system (ex-set-1)

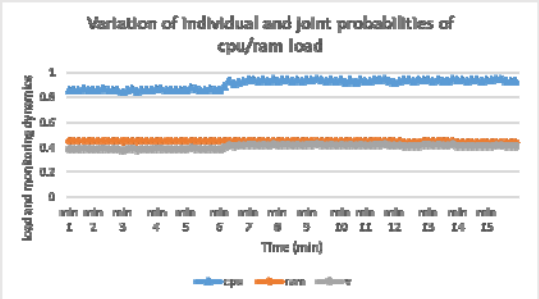

Figure 13. Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-1)

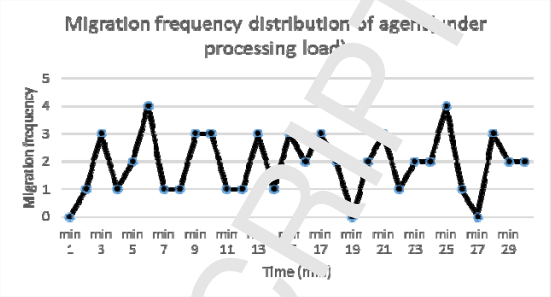

Figure 14. Migration frequency distribution of agent under cpu_load (ex-set-1)

6.2. Set 2: Experimenting with CPU utilization in terms of Heavy Load and video on demand (VOD) Benchmarks

In this case, the cpu_load and video on demand (VOD) benchmarks are employed for estimating the behavior of zone dynamics, rule firing density, combined dynamics of CPU/RAM utilization, joint probability and migration frequency. The variation in zone dynamics with respect to increasing cpu_load and VOD is illustrated in Figure 15. Initially, cpu_load is in zone_1, however 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 Figure 16. The variations in the combined dynamics of CPU/RAM utilization and, joint probability are illustrated in Figure 17. Under increased cpu_load and VOD, the status of CPU utilization becomes maximum. The variations in migration frequency are illustrated in Figure 18. It is shown in the figure that increasing the CPU load and VOD causes frequent migration of agent between different zones.

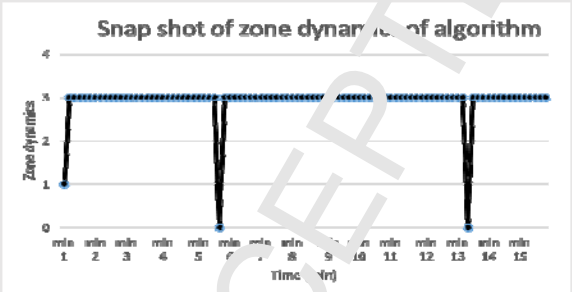

Figure 15. Zone dynamics of mobile agent monitoring system (ex-set-2)

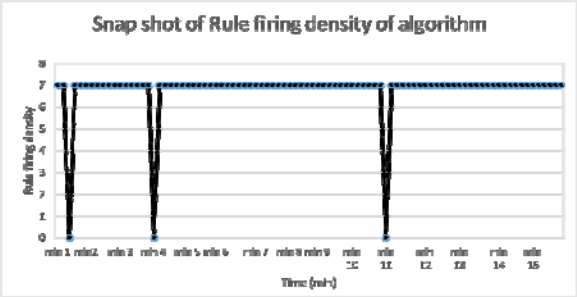

Figure 16. Rule firing density of mobile agent monitoring system (ex-set-2)

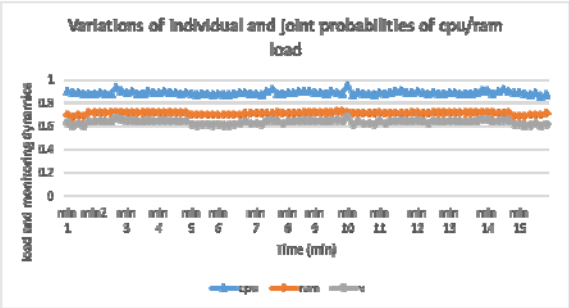

Figure 17. Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-2)

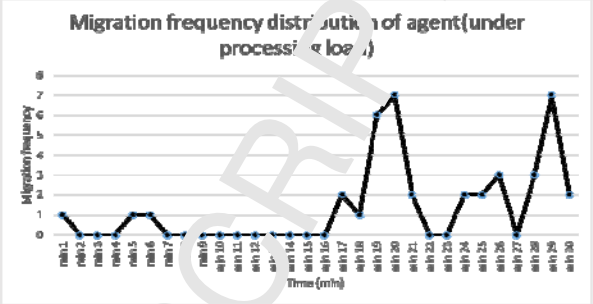

Figure 18. Migration frequency distribution of agent under cpu_load and VOD load (ex-set-2)

6.3. Set 3: Experimenting with CPU utilization in terms of Heavy Load, CPU Stress and video on demand (VOD) Benchmarks

In this case, the cpu_load is employed by applying heavy load generation software, VOD and CPU stress software for estimating the behavior of zone dynamics, rule firing dynamics, combined dynamics of CPU/RAM, joint probability and, migration frequency. The variation in zone dynamics with respect to increasing cpu_load is illustrated in Figure 19. The variations in rule firing density are illustrated in Figure 20. The variations in the combined dynamics of CPU/RAM utilization and joint probability are illustrated in Figure 21. The variations in migration frequency are illustrated in Figure 22.

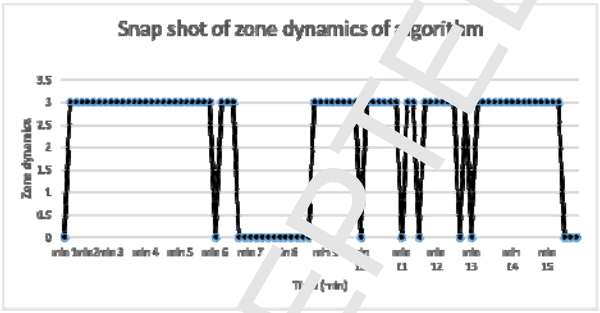

Figure 19. Zone dynamics of mobile agent monitoring system (ex-set-3)

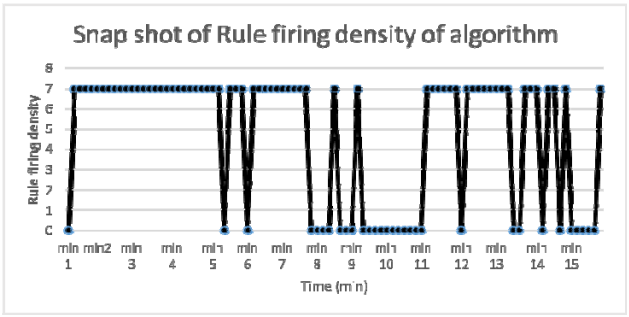

Figure 20. Rule firing density of mobile agent monitoring system (ex-set-3)

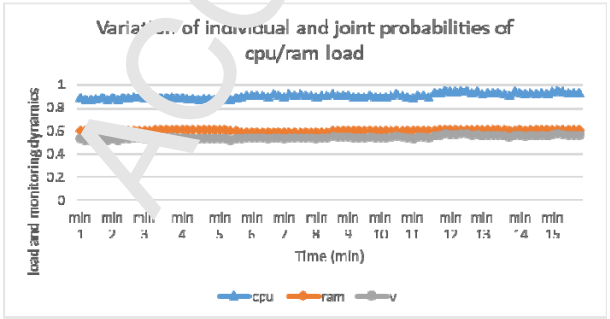

Figure 21. Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-3)

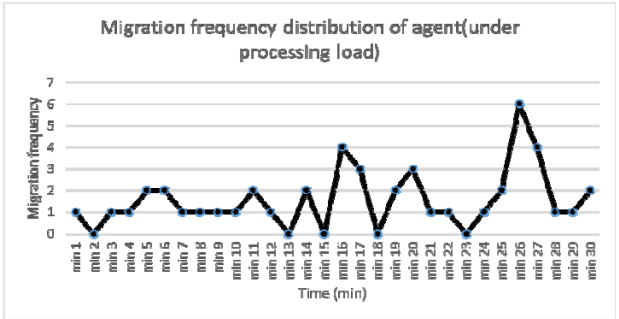

Figure 22. Migration frequency distribution of agent under cpu_load and VOD load (ex-set-3)

6.4. Set 4: Experimenting with CPU and RAM utilization in terms of Heavy Load and CPU Stress Benchmarks

In this case, the cpu_load and ram_load are employed by applying heavy load generation software, VOD and CPU stress software for estimating the behavior of zone dynamics, rule firing dynamics, combined dynamics of CPU/RAM utilization, joint probability and, migration frequency of our proposed algorithmic approach. The variation in zone dynamics with respect to increasing cpu_load is illustrated in Figure 23. The variations in rule firing density are illustrated in Figure 24. The variations in the combined dynamics of CPU/RAM utilization and, computed joint probability are illustrated in Figure 25. The variations in migration frequency are illustrated in Figure 26.

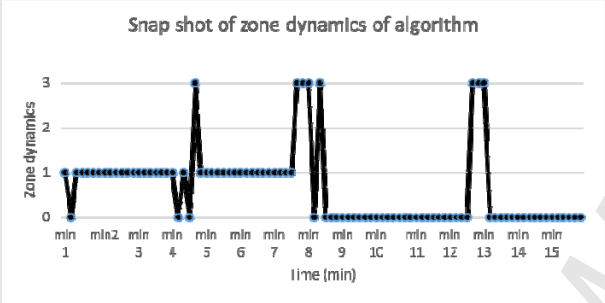

Figure 23. Zone dynamics of mobile agent monitoring system (ex-set-4)

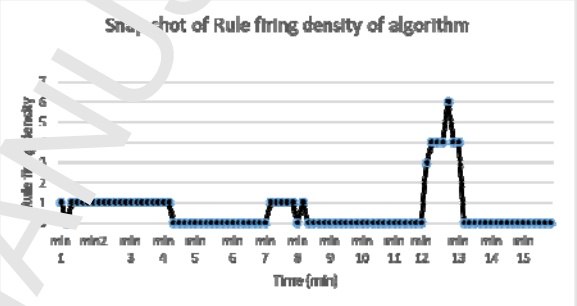

Figure 24. Rule firing density of mobile agent monitoring system (ex-set-4)

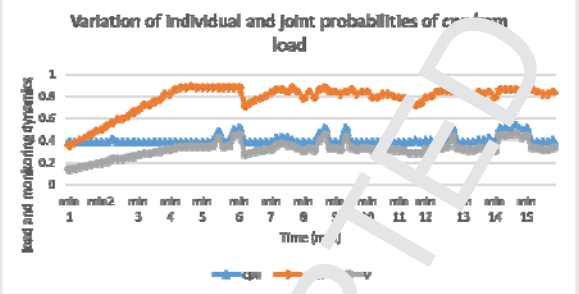

Figure 25. Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-4)

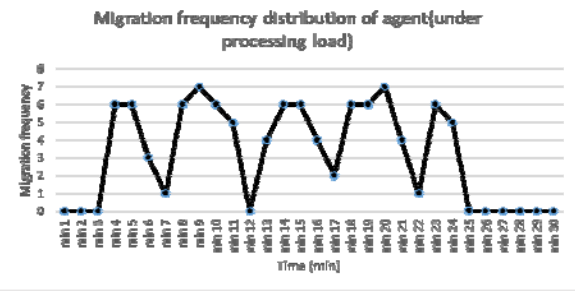

Figure 26. Migration frequency distribution of agent under cpu_load and VOD load (ex-set-4)

6.5. Set 5: Experimenting with RAM utilization in terms of Heavy Load and video on demand (VOD) Benchmarks

In this case, the cpu_load and ram_load is employed by applying heavy load generation software and VOD for estimating the behavior of zone dynamics, rule firing dynamics, combined dynamics of CPU/RAM utilization, joint probability and, migration frequency. The variation in zone dynamics with respect to increasing cpu_load is illustrated in Figure 27. The variations in rule firing density are illustrated in Figure 28. The variations in the combined dynamics of

CPU/RAM utilization and, joint probability are illustrated in Figure 29. The variations in migration frequency are illustrated in Figure 30.

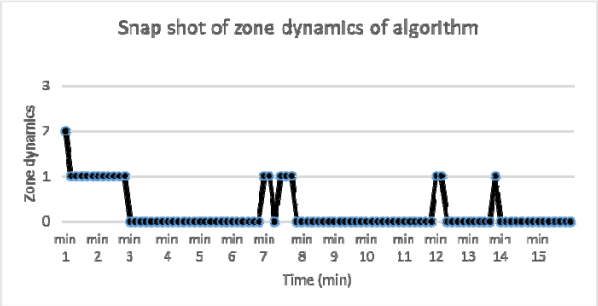

Figure 27. Zone dynamics of mobile agent monitoring system (ex-set-5)

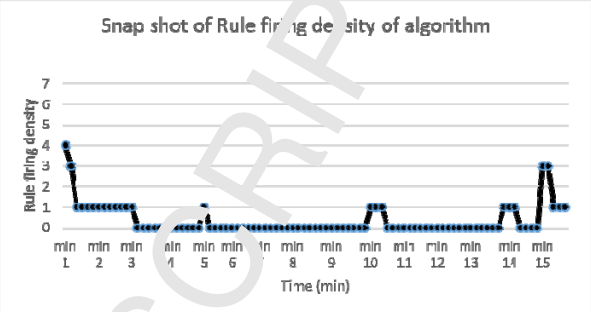

Figure 28. Rule firing density of mobile agent monitoring system (ex-set-5)

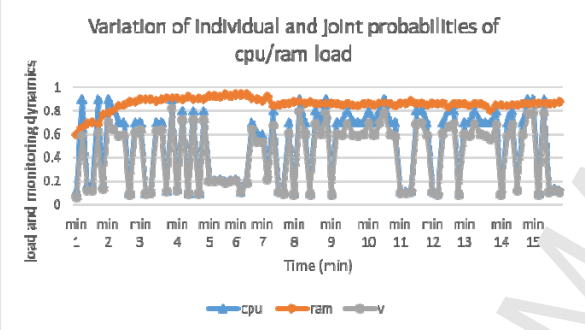

Figure 29. Combined dynamics of CPU and RAM load of mobile agent monitoring system (ex-set-5)

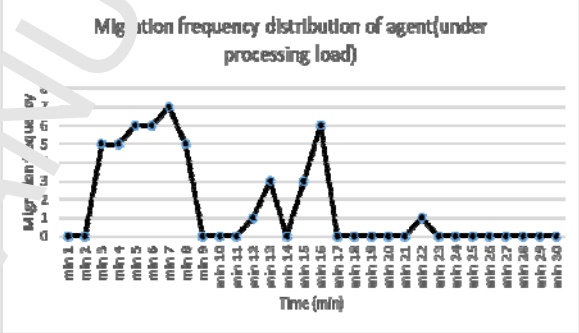

Figure 30. Migration frequency distribution of agent under cpu_load and VOD load (ex-set-5)

6.6. FFT-z and DGEMM Stress Test

In this section, the experimental setup and performance of the monitoring algorithm are described by using resource utilization and decision logic execution. We have conducted four set of experiments (ex-1 to ex-4) under different load distribution in order to evaluate the decision logic execution generated by monitoring agents. Moreover, we have evaluated the decision logic execution by monitoring agent using a set of numerically tagged decision logic parameters such as, 1: *any_load*, 2: *compute_cpu_ram_load*, 3: *light_cpu_load*, 4: *light_ram_load*, 5: *light_cpu_ram_load*, 6: *migrate_agent*, 7: *ram_load*, 8: *cpu_load* and, 9: *stop_sending_load*. In the first experiment, the CPU and RAM utilization is evaluated using FFT-z CPU stress benchmark having CPU cores saturation as core-C0, core-C2, core-C4 and, core-C6. The variations in resource utilization and decision logic execution for this experiment are illustrated in Figure 31 and Figure 32 respectively. It is evident that the behavior of decision logic execution is periodic in nature. The reason for periodic execution behavior is due to the current load having average ranged values (CPU 50% and RAM 60%). In the second experiment, the saturation of CPU cores is set to core-C0, core-C1, core-C3, core-C4 and, core-C6 as illustrated in Figure 33. The behavior of decision logic execution is aperiodic in nature. The reason for aperiodicity is due to the algorithmic logic of monitoring

algorithm to execute various decision outputs as illustrated in Figure 34. In the third experiment, the saturation of CPU cores is set as core-C0, core-C1, core-C2, core-C3, core-C4, core-C5, core-C6 and, core-C7 as illustrated in Figure 35. In this case, the average value of resource utilization of CPU is 100% and utilization of the RAM is 60 %. The reason for full utilization of CPU is due to saturating maximum available number of activated CPU cores. The decision logic execution of monitoring agent is periodic in nature due to full utilization of CPU exceeding the specified threshold value. Therefore, as illustrated in the Figure 36 the decision logic 9 (stop_sending_load) is validated. In the fourth experiment, the stress test is carried out by the DGEMM CPU stress benchmark as illustrated in Figure 37. The DGEMM is a stress test model based on multithreaded programming. The sets of matrix dimensions for DGEMM are consisting of 150x150, 500x500, 1000x1000 and 1500x1500. The average resource utilization of CPU is 40% while the RAM utilization is 70% on the average. The decision logic execution sequence of agent monitoring algorithm is illustrated in Figure 38. In this case, the mobile agent executes decision logic 5 (*light_cpu_ram_load*) and 8 (*cpu_load*) in most cases. The reason for frequent execution of decision logic 5 and decision logic 8 is due to the current lightly loaded status of a node. Therefore, the monitoring node can send CPU load as well as light RAM load to a particular node.

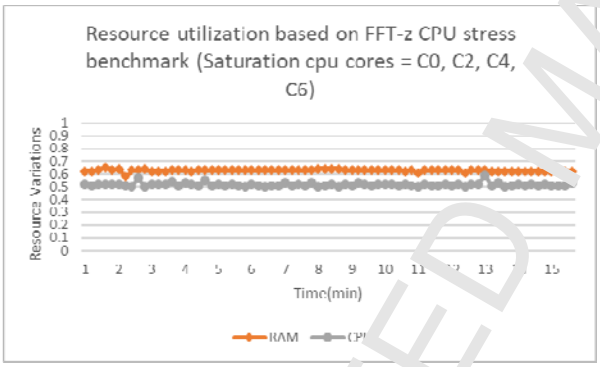

Figure 31. Resource utilization of mobile agent monitoring system (ex-1)

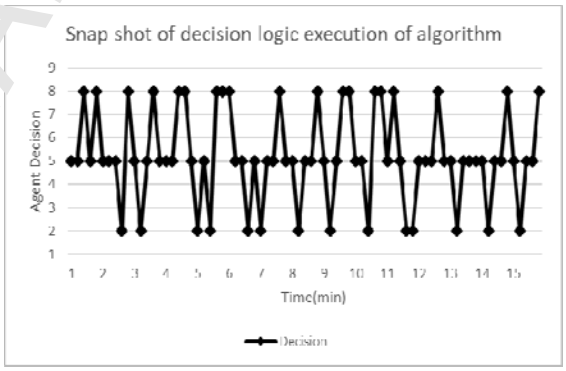

Figure 32. Decision logic execution of mobile agent monitoring system (ex-1)

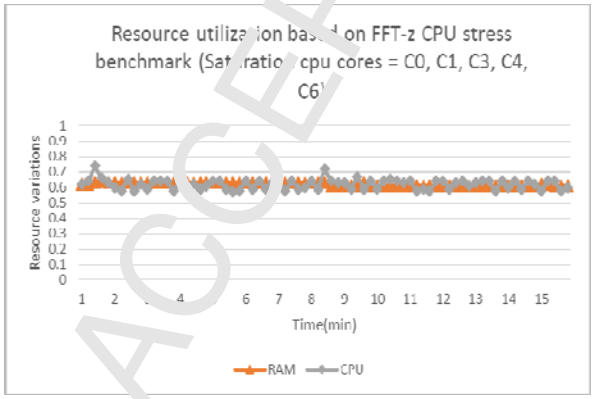

Figure 33. Resource utilization of mobile agent monitoring system (ex-2)

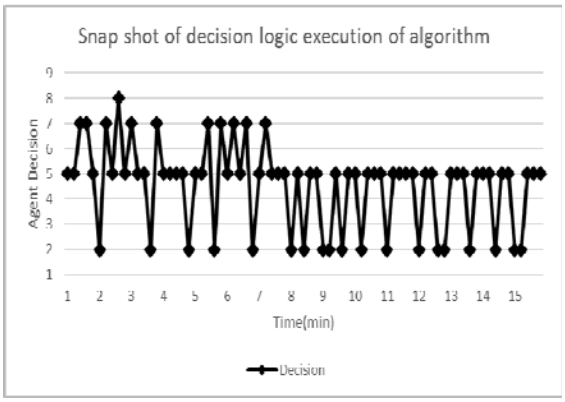

Figure 34. Decision logic execution of mobile agent monitoring system (ex-2)

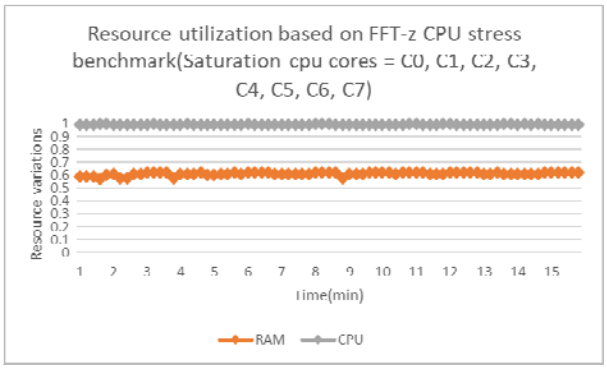

Figure 35. Resource utilization of mobile agent monitoring system (ex-3)

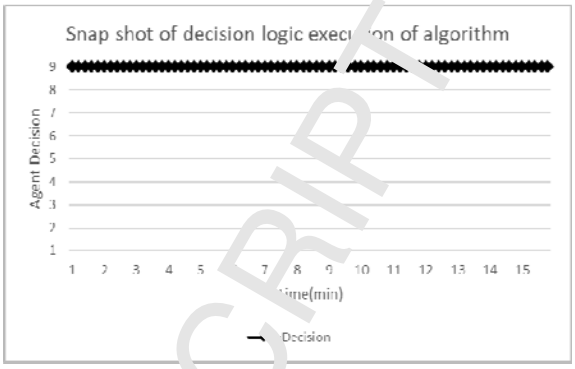

Figure 36. Decision logic execution of mobile agent monitoring system (ex-3)

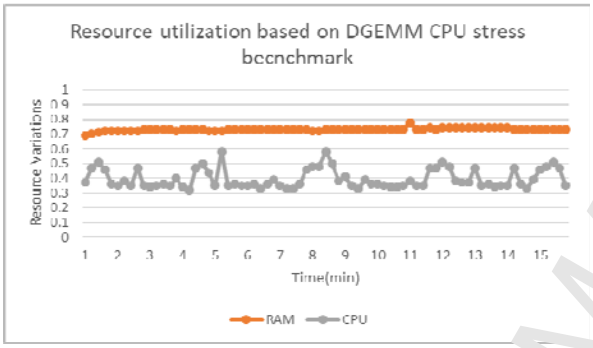

Figure 37. Resource utilization of mobile agent monitoring system (ex-4)

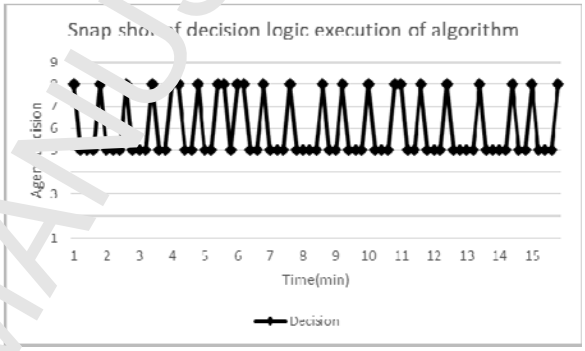

Figure 38. Decision logic execution of mobile agent monitoring system (ex-4)

6.7. Decision Execution Profile of Agents

In our monitoring algorithm, decision execution block of our mobile agent is designed to fire decision sequences according to the computing status of currently available resources at a node at an instant of time. To evaluate the decision sequences generated by agents, we have conducted five experiments (ex-1 to ex-5) under different load conditions. We have evaluated the decision sequences by using a set of numerically tagged decision logic given by, 1: *any_load*, 2: *compute_cpu_ram_load*, 3: *light_cpu_load*, 4: *light_ram_load*, 5: *light_cpu_ram_load*, 6: *migrate_agent*, 7: *ram_load*, 8: *cpu_load* and, 9: *stop_sending_load*. The decision execution profile represents the sequence of fired decision logic by the mobile agent in the specific time interval. In the first experiment, the dynamism of CPU utilization is evaluated in terms of Heavy load and CPU stress benchmarks. The variation in decision execution sequences for experiment one is illustrated in Figure 39 (the x-axis represents time and y-axis represent numerically tagged decision logic). The behavior of decision execution sequence is aperiodic, because the current load status of the node is less than the threshold value. In the second experiment, the dynamism of CPU utilization is evaluated in terms of Heavy load and video on demand (VOD) benchmarks. The variation in decision execution sequences for the second experiment is illustrated in Figure 40. The behavior of decision logic sequence is aperiodic because of the gradual increase in the CPU load. In the third

experiment, the dynamism of CPU utilization is evaluated in terms of Heavy load, CPU stress and, VOD combined benchmarks. The variation in decision execution sequences for the third experiment is illustrated in Figure 41. In this case, the mobile agent initially fires decision logic 5 (*light_cpu_ram_load*) based on current load status. However, further increased in load results in firing decision logic 9 (*stop_sending_load*), because the load is now increased from the specified threshold value in the current node. In the fourth experiment, the dynamism of CPU and RAM utilizations are evaluated in terms of Heavy load and CPU stress benchmarks. The variation in decision execution sequences for the fourth experiment with respect to increasing CPU and RAM load is illustrated in Figure 42. In the fifth experiment, the dynamism of RAM utilization is evaluated in terms of Heavy load and VOD benchmarks. The variation in decision execution sequences for the fifth experiment with respect to increasing RAM load is illustrated in Figure 43. In this case, the mobile agent initially executes decision logic 1 (*any_load*), which means that the current load status of the node is marked as very lightly loaded and the monitoring node can send any load to this particular node. After more than halftime of the conducting experiment, the mobile agent executes decision logic 2 (*compute_cpu_ram_load*) and 8 (*cpu_load*). However, the overall decision logic sequence remains stable at sequence 1 (*any_load*) for a relatively long period of time, because of the lightly loaded status of the respective node.

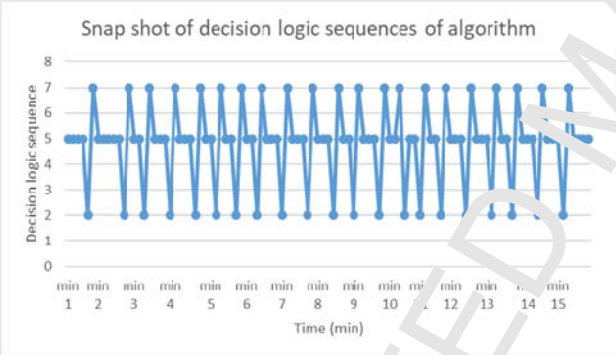

Figure 39. Decision logic sequence of mobile agent monitoring system (ex-1)

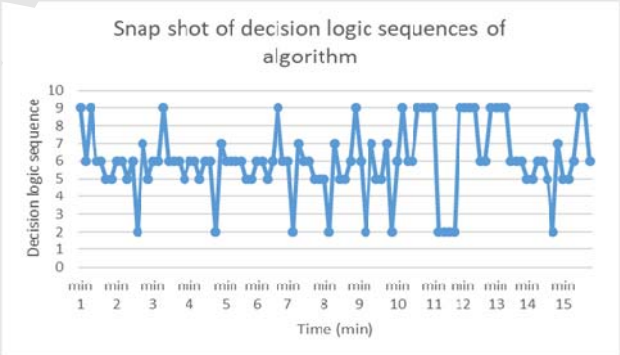

Figure 40. Decision logic sequence of mobile agent monitoring system (ex-2)

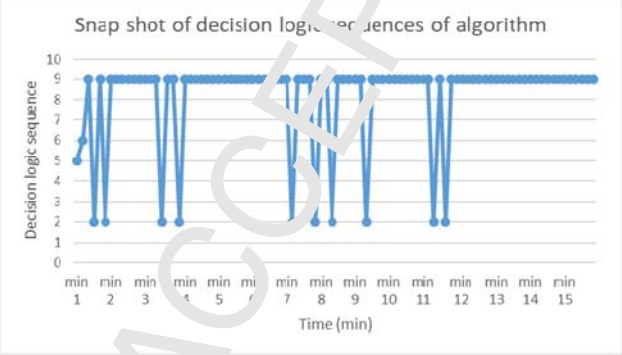

Figure 41. Decision logic sequence of mobile agent monitoring system (ex-3)

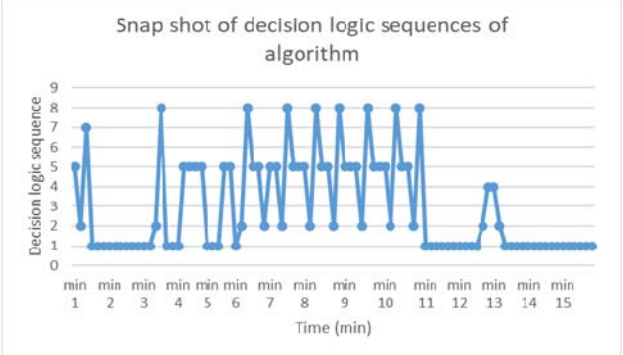

Figure 42. Decision logic sequence of mobile agent monitoring system (ex-4)

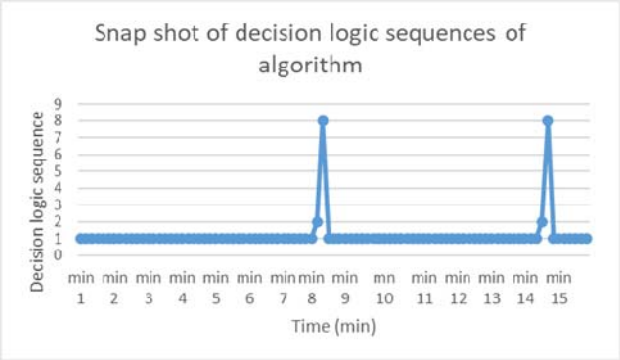

Figure 43. Decision logic sequence of mobile agent monitoring system (ex-5)

6.8. Scalability estimation

Scalability is the ability of a system, a network or an application to maintain overall performance at an acceptable level even if the load is increased. In this section, we have conducted experiments to evaluate the scalability of the proposed design architecture by computing response time of mobile agents. Our experimental testbed is consisting of a group of nine computing nodes at maximum, which are connected by heterogeneous networks (wired and wireless networks). We have considered heterogeneous hardware platforms to prepare testbed, which are comprised of: (a) nodes having dual-core Celeron 2.8GHz CPU and limited RAM (4GB) and, (b) nodes having core i7 3.4GHz (onboard 8 CPUs) and 8GB RAM. We have measured response time as an average of three repeated experiments on the testbed for each node counts. We have conducted scalability analysis in two phases such as, (1) experimentally measuring on the testbed and, (2) theoretically projecting through regression analysis for a very large set of nodes based on our experimental data.

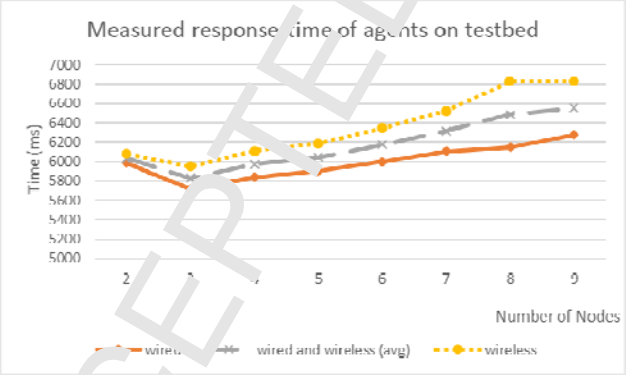

Figure 44. Measuring response time of agents on testbed

In Figure 44, experimentally measured response time is presented for wired network, wireless network and hybrid network (wired and wireless). It is illustrated in Figure 44 that, the average response time of nodes connected by a wired network is less than wireless connected nodes. This is because the available bandwidth of the wired network is greater than the wireless network. Moreover, the wired medium is less prone to data losses and crosstalk as compared to wireless

medium. In order to analyze the behavior of our proposed model for a very large set of nodes, we have performed a regression analysis for projection based on Equation 5.

$$y_i = b_0 + b_1(x_i) \quad (5)$$

The values of coefficients b_0 and b_1 are computed by using the experimental data of Figure 44. The values of b_0 and b_1 are given in Table 2.

Table 2. Computed coefficient values

Wired Nodes	
b_0	4416.85
b_1	247.75
Wireless Nodes	
b_0	446[illegible]
b_1	[illegible]

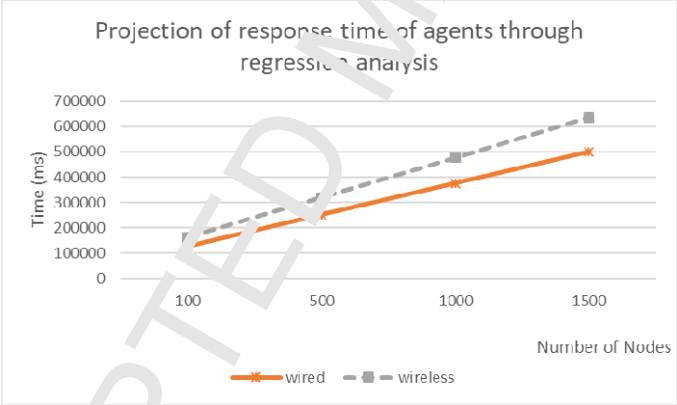

Figure 45. Projection of response time of agent through regression analysis

The projection of response time of mobile agents for a very large set of nodes is illustrated in Figure 45. The regression analysis shows that, for a very large set of nodes, the response time of agents tends to diverge considering wired and wireless networks. The wired network connected nodes are having better response time due to high bandwidth and less interference in data communication.

7. Comparative Analysis

We have evaluated our mobile agent-based load monitoring model with various agent-based load monitoring frameworks for performance analysis. Some of these include, *Agent-Based Adaptive Monitoring System* [21], *Java Based Agent Management System* [23], *Localhost Information*

Service Agent (LISA) [14], *Web Server Load Monitoring System using a Mobile Agent* [12], *A Method of Network Monitoring by Mobile Agents* [1], *A Mobile Agent-Based System for Server Resource Monitoring* [2] and, *Monitoring Agents in A Large Integrated Services Architecture* (MonALISA) [22]. Each mobile agent model is examined and evaluated with respect to various design parameters and attributes like autonomisity, reliability, mobility, network latency, heterogeneity and, flexibility. A detailed discussion is explained in the following sections.

7.1. Qualitative Analysis

In our qualitative analysis, we have selected six parameters for qualitative analysis namely, autonomousity, reliability, mobility, network latency, flexibility and, heterogeneity as illustrated in Table 3. In the qualitative analysis, the parameters are categorized in between [0, 1] scale, where 0 indicates the low value and 1 indicates high value. Moreover, within 0 and 1 scale we have categorized the interval into three zones (first zone is from 0 to 0.3 which indicates low zone (L), the second zone is from 0.3 to 0.6 indicating medium zone (M) and, the last zone is from 0.6 to 1 indicating high zone (H)).

7.1.1. Autonomousity

In case of autonomousity, the agent-based adaptive monitoring system (ABAMS) is in high zone, because the system administrator does not need to update the system information when new nodes are appended (the information updating is carried out by the mobile agent) [21]. The autonomousity of mobile agent based network monitoring and management system (MABNMMS) is in high zone, because the agent will migrate to various nodes in the network, gather status information and carry out device control tasks [1]. This model is designed for decentralized networks without autonomousity. The autonomousity of Monitoring Agents in A Large Integrated Services Architecture (MonALISA) is in a high zone, because this model is designed as an autonomous, multi-threaded agent-based monitoring system. MonALISA registers dynamic services which are able to collaborate and cooperate for performing a wide range of monitoring tasks in distributed systems [22]. Without autonomous behavior, the collaborating and cooperating with other agents will be difficult to achieve. The autonomousity of Mobile Agent-Based Server Resource Monitoring System (MABSRMS) is in high zone, because this is designed to collect available system resources in the large scale-distributed systems [12]. This model uses three different mobile agents to collect system resources. The autonomousity of Distributed Management System based on Java agents (DMSBA) is in the medium zone, because the management is handled by the administrator, but when the system loses contact with central administrator or having problems on time scales, the mobile agent will take over and autonomously manages the system and performs automated tasks [23]. The autonomousity of Mobile Agent-Based Load Monitoring (MABLM) is in the high zone because the agent is designed in such a way that it autonomously and automatically gathers system status information and sends the results to monitoring server [20]. The autonomousity of Localhost Information Service Agent (LISA) is in high zone, because this monitoring system is designed to dynamically configure system parameters and autonomously migrates agent to various nodes in the network to increase the system performance [14]. The autonomousity of our mobile agent-based monitoring system is in a high zone, because the mobile agent collects system status

information automatically. In our model, the agent must be autonomous, because the agent will make a decision about migrating to the next node. If the node is not available then the agent will select the next available node in the network.

7.1.2. Reliability

Reliability is used to measure the system stability. The reliability of agent-based adaptive monitoring system (ABAMS) is in the low zone, because server broker is used for collecting information of active nodes by sending heartbeat signals [21]. The reliability of mobile agent based network monitoring and management system (MABNMMS) is in high zone, because in this model fault management module is used to monitor faults and take necessary actions accordingly to increase overall system reliability [1]. The reliability of the MonALISA monitoring system is in a high zone, because this model uses a dynamic pool of threads [22]. The reliability of Mobile Agent-Based Server Resource Monitoring System (MABSRMS) is in the medium zone, because it uses more than one server for enhancing reliability [12]. The reliability of Distributed Management System based on Java agents (DMSBA) is in high zone, because this model uses mobile agents that can be trained to perform a set of tasks such as, monitoring host network traffic and restarting a specific server that has been crashed recently to achieve a high level of reliability [23]. The reliability of Mobile Agent Based Load Monitoring (MABLM) is in the medium zone, because this model uses master-slave agents for monitoring [20]. The master agent keeps track of slave agents in the system. The reliability of Localhost Information Service Agent (LISA) is in high zone, because this framework uses the binary module, which can be automatically restarted by the mobile agent creating reliable monitoring system [14]. The reliability of our mobile agent-based monitoring system is in a high zone, because our mobile agent checks the status of each node. If some node goes offline then the agent will immediately notify the monitoring node and the monitoring node will take necessary action accordingly.

7.1.3. Mobility

In our qualitative analysis, we have selected seven models which are based on mobile agents. All these models are in the high zone in terms of mobility, because mobility is an essential factor of agents.

7.1.4. Network Latency

In the case of network latency ABAMS model is in a high zone, because the agent will communicate with server broker, not with monitoring system directly [21]. The communication between server broker and monitoring node will increase intercommunication, which in turn increases the network latency. Network latency of MABNMMS monitoring system is in the medium zone, because the mobile agent collects status information of all available nodes in the network and sends it to the manager node [1]. The mobile agent does not send individual node status information, which increases the delay in the monitoring process. Network latency of MonALISA monitoring model is in the low zone, because the reusability of the dynamic pool of threads reduces the network load as well as network latency increasing overall system performance [22]. Network latency of MABSRMS model is in the medium zone, because this model uses three different agents for collecting system information, which increases the network

latency. Network latency of the DMSBA model is in the medium zone, because this model replicates data on multiple servers for reliability, which increases the network latency [23]. Network latency of MABLM model is in the medium zone, because the master agent will create slave agents and then it migrates slave agents to collect system information for available nodes in the network [20]. Network latency of LISA monitoring model is in the low zone, because LISA monitoring system can dynamically load or unload its binary modules to monitor current status and sends results directly to a monitoring system [14]. Moreover, network latency of our mobile agent-based monitoring system is in the low zone, because the mobile agent collects status information and directly sends to the monitoring node.

7.1.5. Flexibility

In the case of flexibility, ABAMS model is in a high zone, because each monitoring component can be dynamically activated, deactivated, moved or changed without affecting or restarting the entire monitoring system [21]. The flexibility of MABNMMS model is in the low zone, because the module or component cannot be changed once the system is deployed [1]. The flexibility of MonALISA monitoring model is in the medium zone, because this framework allows co-operating services [22]. The flexibility of MABSRMS model is in the low zone. The reason is that, this model has no mechanism to adopt changes in its monitoring component [12]. The flexibility of the DMSBA model is in a high zone, because this model has the capability to dynamically load new code to existing code for enhancing agent capabilities and functionality [23]. The flexibility of MABLM monitoring model is in the low zone, because once the model has deployed, there no mechanism to upgrade its existing components [20]. The flexibility of the LISA monitoring model is in a high zone, because LISA framework is flexible enough to include new modules to the existing system without compromising the performance and efficiency [14]. The flexibility of our monitoring model is in the medium zone, because we can upgrade its monitoring functionality before deployment, but once it is deployed we cannot change its component.

7.1.6. Heterogeneity

The heterogeneity of ABAMS model is in a high zone, because this model uses dynamically controllable agents in a distributed network to utilize heterogeneous resources efficiently. This model uses an agent which can be loaded, unloaded on a specific resource or it can be migrated to another node depending on the requirement. Heterogeneity of MABNMMS monitoring framework is in a high zone, because it can be deployed on various nodes having a different platform in a distributed system. Heterogeneity of MonALISA is in high zone, because this model allows cooperating services and adaptation to a dynamic environment. Moreover, cooperating services also make this monitoring framework very efficient when monitoring a large number of heterogeneous nodes having different response times. The heterogeneity of the DMSBA model is in a high zone, because the interoperability of Java allows this architecture to be highly adaptable and, portable. The heterogeneity of MABLM monitoring model is in high zone, because this monitoring framework uses Mobile Agent Server (MAS), which provides an

environment for executing mobile agents on various nodes (heterogeneous nodes) in distributed systems. Heterogeneity of LISA framework heterogeneity is in high zone. This heterogeneity is managed by the ability to detect various architectures and deploying its binary modules dynamically to monitor its current status. Heterogeneity of our mobile agent monitoring system is in a high zone, because our monitoring model is developed using Java language, which provides interoperability and can be easily deployed in heterogeneous platforms.

Table 3. 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

Legends: 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).

7.2. Quantitative Analysis

In this section, we will quantitatively analyze different agent-based monitoring models. The comparative studies are performed based on fault tolerance, scalability, mobile agent migration latency and, response time as illustrated in Figure 46, 47, 48, 49 and, 50, respectively. The values of performance metric are determined with approximation by analysis. The metric values are set in 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).

7.2.1. Scalability

Scalability is defined as the ability of a system or model that describes its capability to perform efficiently under an increasing number of nodes. The scalability of ABAMS framework is in a high zone, because it can monitor more than one multicast groups, however, the administrator can add or remove multicast group without disturbing the system performance [21]. MABNMMS monitoring system has scalability in the medium zone, because this framework is tested on a fixed number of nodes [1]. Scalability of MonALISA framework is in the medium zone, because it is designed as an autonomous multi-threaded self-describing agent-based system, which is able to collaborate and cooperate in performing various monitoring task by adding more nodes in the existing system without affecting system performance [22]. Moreover, the scalability of MABSRMS monitoring framework is in the medium zone, because it is deployed in large-scale distributed systems. The mobile agent calls low-level functions in dynamic libraries and can monitor system resources efficiently without considering the increased number of nodes [12]. The scalability of the DMSBA monitoring system is in the medium zone. It is designed for large-scale distributed systems [23]. The scalability of MABLM is in high zone, because it is designed for large-scale distributed system and operates efficiently where a large number of systems are connected through a network having low bandwidth and high delay time [20]. The scalability of LISA framework is in high zone, because the agents automatically detect the architecture and load binary modules (in case of the appended node) necessary to perform monitoring services making this framework scalable [14]. The scalability of our mobile agent monitoring system is in a high zone, because the mobile agent detects any addition or removal of nodes in the system and updates its information in the node information database.

7.2.2. Fault Tolerance

In case of fault tolerant, ABAMS framework is in the low zone, because there is no mechanism specified which can be used to recover from the failure [21]. For example, if the monitoring agents die or stop working due to some error, then the entire system fails. The fault tolerance of MABNMMS monitoring system is in a high zone, because the fault management module uses a mobile agent to monitor the network and identify any faults and take a necessary control action [1]. The fault tolerance of MonALISA framework is in high zone, because this framework assigns an independent thread to each task so that if some task fails due to error other tasks should not be affected which make this framework fault tolerant [22]. The fault tolerance of MABSRMS monitoring system is in the low zone, because there is no backup mechanism in monitoring system [12]. The fault tolerance of DMSBA framework is in a high zone, because the agents detect failures or congestion automatically and, change the granularity at which the data is collected [23]. This mechanism provides the administrator with a detailed view of the system. The fault tolerance of MABLM monitoring system is in the medium zone. The agents communicate with each other if some fault or error occurs and agents notify the monitoring system while taking control action enhancing fault tolerance [20]. The fault tolerance of LISA monitoring system is in a high zone, because the core system manages monitoring modules and dynamically restarts the crashed module [14]. The fault tolerance of our mobile agent-based

monitoring system is in a high zone, because the mobile agent migrates to each node and collects status information. If any node dies or an error occurs, then the mobile agent immediately informs the monitoring system. The mobile agent constantly updates the node information database, which enhances overall system fault tolerance.

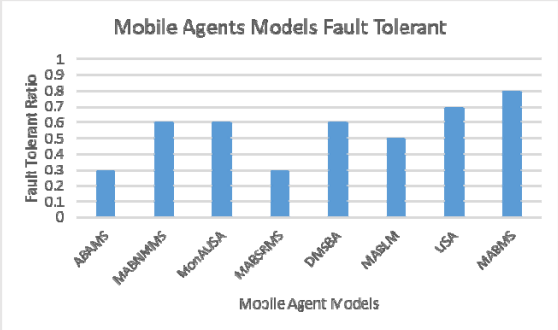

Figure 46. Fault tolerance of mobile agent models

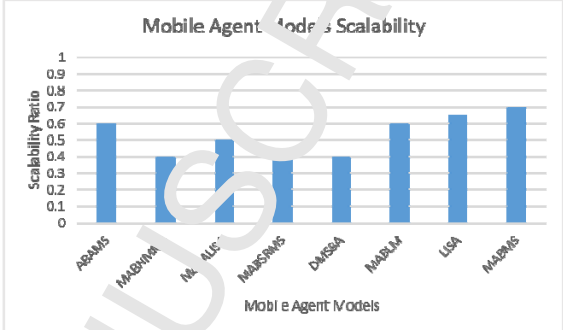

Figure 47. Scalability distribution of mobile agent models

7.2.3. Mobile Agent Migration Latency

In Figures 48 and 49, the comparison of mobile agent migration latency (excluding agent code execution time at nodes) is illustrated for the wired and wireless nodes for varying data sizes of agents. As illustrated in Figure 48 and 49, 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 Figure 48 and 49, 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.

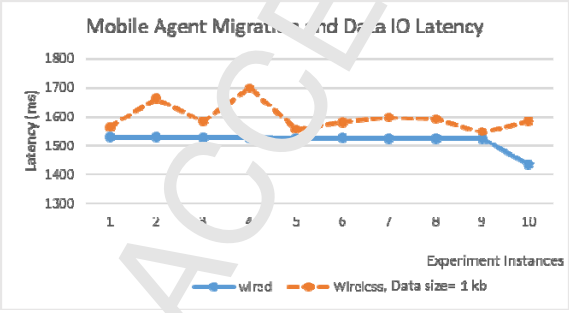

Figure 48. Comparison of mobile agent migration and data IO latency with respect to wired and wireless node (data size = 1kb)

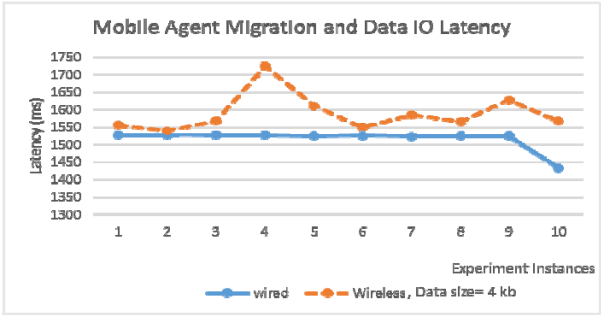

Figure 49. Comparison of mobile agent migration and data IO latency with respect to wired and wireless node (data size = 4 kb)

7.2.4. Execution-response Time

In Figure 50, the comparison of execution-response time (i.e. averaged value of agent code execution time at nodes excluding network IO and migration time) for the 3-node model is illustrated considering MABNMMS and MABMS. As illustrated in Figure 50, the execution-response time of MABMS is less than MABNMMS, which shows that MABMS performance is better than MABNMMS. The MABNMMS monitoring agents use a shell script [1]. The results illustrate that, the execution-response time depends upon the complexity of the script as well as the number of available nodes in the network. Execution-response time increases with the increase in a number of nodes in the network and, vice versa. While MABMS uses Java-based agents, the Java-based agents are platform independent and lightweight in nature. This reduces the execution-response time of agents in MABMS based on JADE platform.

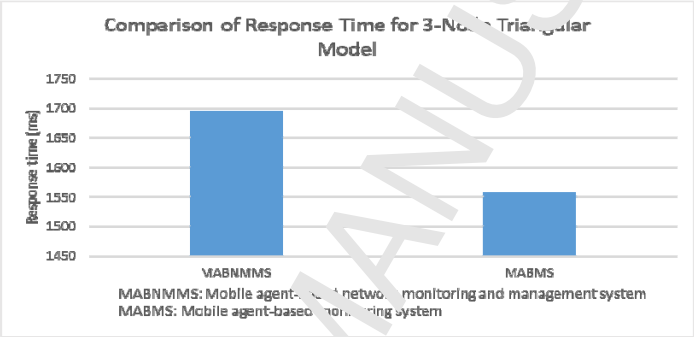

Figure 50. Comparison of response time for 3-Node triangular model

8. Conclusion

In large-scale distributed systems, keeping track of load monitoring mechanism by a system administrator is a very difficult task. Load monitoring mechanism is used to monitor available computing nodes in a distributed system. In this paper, we present the design, implementation, and evaluation of mobile agents based load monitoring system for distributed systems in a heterogeneous network environment. Moreover, the monitoring algorithm runs to compute the status of currently available node resources in time intervals. We used mobile agents to collect status information from nodes and sending a response message to the monitoring node. The pseudocodes describing monitoring algorithms and decision algorithm are presented. In order to evaluate the behavior and performance of our implemented system, we have conducted five sets of experiments under different load scenarios. We have evaluated the performance using a set of parameters such as, zone dynamics, rule firing density, load dynamics and, migration frequency under different load conditions. A detailed scalability analysis is performed based on experimental evaluation on testbed and through regression analysis. The proposed approach reduces the waiting time of a node as well as the network load in order to increase the overall system performance. The waiting time of a node is reduced by autonomously sending the required information to the monitoring node for decision making at different time instants.

References

[1] Manvi, S, S and Venkataram, p. A method of network monitoring by mobile agents, computing, 2000, 2(3), pp. 4-5.

[2] 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.

[3] Das, Shantanu., Mobile agents in distributed computing: Network exploration. Bulletin of EATCS 1, no. 109, 2013.

[4] Park, H. J., K. J. Jyung., and S. S. Kim., Mobile agent-based load monitoring system for the safety web server environment. In International Conference on Computational Science, Springer, 2004, pp. 274-280.

[5] 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.

[6] Xu F., Liu F., Liu L., Jin H., and Li B., iAware: Making live migration of virtual machines interference-aware in the cloud, IEEE Transactions on Computers, volume: 63, IEEE, 2014, pp. 3012-3025.

[7] 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.

[8] Haverkamp DS., and Gauch S. Intelligent information agents: review and challenges for distributed information sources, Journal of the Association for Information Science and Technology 49, no. 4, 1998, page(s) 304-311.

[9] Wang, Xiaoguang, Hui Wang, and Yongbin Wang. A unified monitoring framework for distributed environment. Intelligent Information Management 2, no. 07, 2010, pp. 398.

[10] Massie, Matthew L., Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing 30, no. 7, 2004, pp. 817-840.

[11] Vidhate, Sonali L., and M. U. Kharat. Resource Aware Monitoring in Distributed System using Tabu Search Algorithm. International Journal of Computer Applications 96, no. 23 2014.

[12] Tian, Z. A Mobile Agent-Based System for Server Resource Monitoring. Cybernetics and Information Technologies, 13(4), 2013, pp. 104-117.

[13] Wörn, H., Längle, T., Albert, M., Kazi, A., Brighenti, A., Seijo, S.R., Senior, C., Bobi, M.A.S. and Collado, J.V., Diamond: distributed multi-agent architecture for monitoring and diagnosis. Production planning & control, 2004, 15(2), pp. 189-200.

[14] Dobre, C., Voicu, R., Muraru, A. and Legrand, I.C., A distributed agent based system to control and coordinate large scale data transfers. arXiv preprint arXiv:1106.5171, 2011.

[15] Seenuvasan, P., Kannan, A. and Varalakshmi, P., Agent-Based Resource Management In A Cloud Environment. Appl. Math, 11(3), 2017, pp. 777-788.

[16] Iosup, A., Ţăpuş, N. and Vialle, S., 2005, February. A monitoring architecture for control grids. In European Grid Conference, Springer, 2005, pp. 922-931.

[17] 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.

[18] Gunter, D., Tierney, B., Jackson, K., Lee, J. and Stufer, M., Dynamic monitoring of high-performance distributed applications. In High Performance Distributed Computing, 11th IEEE International Symposium, IEEE, 2002, pp. 163-170.

[19] Hoke, E., Sun, J., and Faloutsos, C., Intemon: Intelligent system monitoring on large clusters. In Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment, ACM, 2006, pp. 1239-1242.

[20] Kim, S.T., Park, H.J. and Kim, Y.C., The load monitoring of Web server using mobile agent. In *Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on* IEEE, Vol. 5, pp. 89-94.

[21] 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.

[22] Legrand, I., Newman, H., Voicu, R., Cirstoiu, C., Grigoras, C., Dobre, C., Muraru, A., Costan, A., Dediu, M. and Stratan, C., MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems. *Computer Physics Communications*, *180*(12), 2009, pp.2472-2498.

[23] Brooks, C., Tierney, B. and Johnston, W., JAVA agents for distributed system management. *LBNL Report,* 1997.

[24] Adacal, M. and Bener, A.B., Mobile web services: A new agent-based framework. *IEEE Internet Computing*, *10*(3), 2006, pp.58-65.

[25] 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.

[26] Du, T.C., Li, E.Y. and Chang, A.P., Mobile agents in distributed network management. *Communications of the ACM*, *46*(7), 2003, pp.127-132.

[27] Ahn, J., Fault-tolerant Mobile Agent-based Monitoring Mechanism for Highly Dynamic Distributed Networks. *IJCSI International Journal of Computer Science Issues*, *7*(3), 2010, pp.1-7.

[28] Papavassiliou, S., Puliafito, A., Tomarchio, O. and Ye, J., Mobile agent-based approach for efficient network management and resource allocation: framework and applications. *IEEE Journal on Selected Areas in Communications*, *20*(4), 2002, pp.858-872.

[29] Ku, H., Luderer, G.W. and Subbiah, B., November. An intelligent mobile agent framework for distributed network management. In *Global Telecommunications Conference, GLOBECOM'97.,1997, IEEE,* Vol. 1, pp. 160-164.

[30] 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.

[31] Iranpour E., Sharifian S., A distributed load balancing and admission control algorithm based on fuzzy type-2 and game theory for large-scale SaaS cloud architectures, Future Generation Computer Systems, Vol. 86, Elsevier, 2018, pp. 81-98.

[32] Volkova, V.N., Chemenkaya, L.V., Desyatirikova, E.N., Hajali, M., Khodar, A. and Osama, A., Load balancing in cloud computing. In *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), IEEE,* 2018, pp. 387-390.

[33] Dasoriya, R., Kotadiya, P., Arya, G., Nayak, P. and Mistry, K., Dynamic load balancing in cloud a data-centric approach. In *Networks & Advances in Computational Technologies (NetACT), IEEE, 2017,* pp. 162-166.

[34] Lastovetsky, A., Szustak, L., & Wyrzykowski, R., Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing. IEEE Transactions on Parallel and Distributed Systems, IEEE, 2017, 28(3), 787-797.

[35] Lastovetsky, A., & Manumachu, R. R., New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters. IEEE Transactions on Parallel and Distributed Systems, IEEE, 2017, 28(4), 1119-1133.

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 and Systems.


Moazam Ali


Susmit Bagchi

- Distributed systems load monitoring using agents and probabilistic norm.
- Adaptive and autonomous load monitoring in distributed systems using agents.
- Mobile agent and probabilistic norm based autonomous load monitoring in distributed systems.