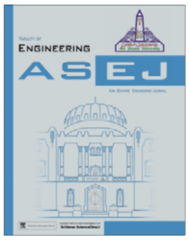
Ain Shams Engineering Journal (2012) 3, 237–249

Ain Shams University

Ain Shams Engineering Journal

www.elsevier.com/locate/asej
www.sciencedirect.com



ELECTRICAL ENGINEERING

Load balancing in distributed multi-agent computing systems

Maha A. Metawei [a,b,*], Salma A. Ghoneim [a], Sahar M. Haggag [a], Salwa M. Nassar [b]

[a] *Computers and Systems Dept., Faculty of Engineering, Ain Shams University, 11517 Cairo, Egypt*
[b] *Electronics Research Institute, 12311 Dokki, Giza, Egypt*

Received 22 January 2012; revised 17 February 2012; accepted 5 March 2012
Available online 23 May 2012

KEYWORDS

Load balancing;
Distributed systems;
Cluster computing;
Multi-agent computing;
Task migration;
Multiple linear regression

Abstract In this paper, a dynamic load-balancing scheme is developed for implementation in an agent-based distributed system. Load balancing is achieved via agent migration from heavily loaded nodes to lightly loaded ones. The credit based concept is used for the dual objective of: (1) the selection of candidate agents for migration, and (2) the selection of destination nodes. This represents an elaboration of previous research work aimed at the selection of agents only. Multiple linear regression is used to achieve our dual objective. A complexity analysis for the proposed system operation is performed, the complete operation is O(n^2) where n is the number of agents on the local host.

The proposed system is implemented using JADE (Java Agent DEvelopment Framework), the multiple linear regression operation is performed using R Tool. The experimental results show a modified system operation in terms of reduced user's query response time, by implementing agent load balancing.

* Corresponding author. Tel.: +20 100 535 8852/022 633 0299.
E-mail addresses: maha_metawei@eri.sci.eg, maha.abdelaziz@gmail.com (M.A. Metawei), salmaghoneim@gmail.com (S.A. Ghoneim), sahar_haggag@eng.asu.edu.eg (S.M. Haggag), salwa@eri.sci.eg (S.M. Nassar).

2090-4479 © 2012 Ain Shams University. Production and hosting by Elsevier B.V. All rights reserved.

Peer review under responsibility of Ain Shams University.
http://dx.doi.org/10.1016/j.asej.2012.03.001

Production and hosting by Elsevier

1. Introduction

Multi-agent systems have recently been widely employed in developing scalable software systems on heterogeneous networks. Indeed, using a cross-platform language, distributed systems based on agents are very attractive because of the inherent scalability and autonomy [1]. A software agent has some special characteristics such as: cooperation, autonomy, ability to communicate, and mobility. Thus a multi-agent system closely resembles a community of human beings doing business with each other while each one of them has one or more predefined goal. Many applications are being built using such a multi-agent paradigm for different purposes. Examples of notable financial multi-agent systems include those reported in [2–4]. A mobile agent-based health care monitoring [5] is designed to notify the responsible care-provider of abnormality automatically.

However, the end-users (e.g., the customer who employs the software agents to locate the cheapest prices of certain commodities in the financial markets or the patient that needs continuous health monitoring) definitely demand a quick response from such a multi-agent system. Because of the unique features (e.g., mobility, autonomy, etc.) of a software agent, load imbalance is inevitable over time and a longer response time results.

Thus, load balancing is needed as a critical system operation to make the multi-agent based distributed computing paradigm attractive. Agent migration is one of the strategies adopted to support load balance in multi-agent systems [6]. In dynamic load-balancing schemes, prior the migration process, two decisions should be made: (1) Which agent should be migrated? (2) Where this agent should go? If the selection policy is formulated carefully, the desirable effects are that the agent selected to migrate will not make the overall situation worse by making the destination more overloaded than the source, and the cost of the migration will be compensated by the gain in performance.

One of the used criteria to select the migrating agent is the credit-based criteria [1]. The system allocates a credit to each agent and decides which agent needs to be migrated using the credit value. Any agent with high credit will be given more chance to keep its current position (the node the agent resides in) with less chance to be selected for migration.

In this paper, a fine grained credit-based selection for the migrating agent and the destination host is proposed. Agent-based and system-based factors are considered in the selection and location equations, those equations' coefficients are calculated through a multiple linear regression analysis [7] as an offline operation before system operation startup.

The rest of the paper is organized as follows. Section 2 provides a survey of related work. Section 3 introduces the proposed load balancing scheme and implementation. Performance evaluation through an experiment is presented in Section 4. Results are displayed in Section 5. Section 6 gives a comparison between the proposed load balancing scheme and other recent schemes. Section 7 concludes the paper with a summary on future work.

2. Related work

Recently, many load-balancing schemes for mobile agent-based system have been proposed. The MAS model [6] is an implementation of a dynamic load balancing scheme for multi-agent system, agent selection is based on agent's credit value, and location selection is based on the inter-machine communication load. The migration decision is taken by a centralized agent that triggers the migration process when it is needed. The centralized control is not suitable for dynamic system as it should collect information more frequently than non-centralized ones, which may overload the network traffic.

Another mobility-based load balancing infrastructure is called JIAC [8]. The migration decision is taken locally which prevents the centralized control problems like (single point of failure, system bottleneck, etc.). A weakness in this system is that the migrating agent selection is made randomly.

A locality-sensitive task allocation and load balancing model is suggested in [9]. It considers the effect of communication among agents as in [1], however this model does not use migration, but it redistributes tasks among agents so as to implement the load balancing of agents. Another task allocation and load balancing scheme is given in [10], the task allocation is also used to implement load balancing. The capacity of a software agent to execute tasks is determined by not only itself but also its contextual agents. Thus the number of allocated tasks on an agent is directly proportional to not only its own resources but also the resources of its interacting agents. If there are large number of tasks queued for a certain agent, the capacities of both the agent itself and its contextual agents to accept new tasks will be reduced.

In [11], a centralized load balancing system is given. The selection policy is based on job's execution time, while location policy is based on negotiation with cluster nodes. A dedicated agent on each node is responsible for the local resource usage accounting, for systems that the user has to pay per usage. The migration decision is based on the comparison with the load threshold value.

The same algorithm is used for other different platforms. It is used for load balancing for web server [12] and load balancing of Mobile Services [13].

In [14], a centralized load balancing system is presented. A new approach is suggested to predict and identify the load condition of the agents. The measured load data are stored to be used in the prediction with exponential averaging. Unnecessary migration of agent is prevented by predicting the change in the load in advance. An effective dynamic load balancing scheme based on the credit concept is also used.

Three basic trends to tackle the problem of load balancing in MAS (Mobile Agent Systems) characterize the recent research in this direction. The first such as [14] corresponds to the situation where some hosts on which agents are running can be heavily loaded while the others are idle or lightly loaded. Secondly, agents can be heavily loaded due to having many tasks [9,10]. Thirdly, some tasks which are executed by agents consume more system's resources and cannot be transferred to any other agent because of causing overloading of agents. A hybrid system is proposed in [15] to solve each type of problem according to the situation, each host has Load Balancing Coordinators which gather the information of its host. By means of the gathered information, they can decide which action is appropriate in order to solve the current problems. While load balancing model in [9,10] focus on problem two, they use task allocation to redistribute the agent load. The prediction-based dynamic load balancing model [14] uses agent migration only to solve the load imbalance problem.

3. Proposed system description

3.1. System description

The proposed load balancing system aims to take advantage of the agent characteristics to create an autonomous system. It also aims to overcome similar systems drawbacks like for example the centralized control which may be a single point of failure and by adding more factors to the selection criteria of the migrating agent and the destination node.

As shown in Fig. 1, the system goes through three main stages: calculating system parameters for the model, information gathering stage then using the calculated model to make the appropriate decisions. In the first stage, a multiple linear regression model is built offline before start up to calculate the parameters of the regression equations for both selection and location policy. The obtained linear models are used by all the nodes through their operations. A multiple linear regression [7] is then performed to learn more about the relationship between several independent or predictor variables and a dependent or criterion variable. The same linear model can be used for several system runs under condition that the agents and systems parameters are the same.

Given a variable y and a number of variables X_1, ..., X_p that may be related to y, then linear regression analysis can be applied to quantify the strength of the relationship between y and the X_j, to assess which X_j may have no relationship with y at all, and to identify which subsets of the X_j contain redundant information about y, thus once one of them is known, the others are no longer informative.

A disturbance term ε_i is added to this assumed relationship to capture the influence of everything else on Y_i other than X_i, …, X_p. A general form for the regression equation is given as:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p + \varepsilon_i \tag{1}$$

where the regressors (the X-s) are also called independent variables or predictor variables. Similarly, regressands (the Y-s) are also called dependent variables, response variables, measured variables, or predicted variables.

There are $p + 1$ unknown parameters $b_0, b_1, \ldots, b_p$. One of the purposes of linear regression is to determine these unknown parameters. The **least-squares estimator** is often used to estimate the coefficients of a **linear regression**. The least-squares estimator optimizes a certain criterion (namely it minimizes the sum of the square of the residuals). The coefficient values reflects the effect of the corresponding independent variable on the output variable, i.e. a negligible coefficient relative to the other coefficients values means that this independent variable has a weak effect on the output. The second stage is a gathering process for the needed agents and system parameters values, those values are used in the next phase to evaluate the obtained equations from phase 1. The third stage is an online evaluation of those policies to select a subset of the local agents to be migrated; it also selects a subset of the system nodes to receive those agents. The framework for load balancing consists of a group of agents where each has a specific role to play and have facility for inter-agent communication [16].

The agents' types can be divided into administrative agents and work agents. Each administrative agent is implemented for managing its local host's load and making migration decision to achieve load balancing, while working agents are responsible for performing the user tasks.

3.2. System policies

The proposed load-balancing model focuses on three policies: information gathering, selection and location policy. Each policy is described separately here after.

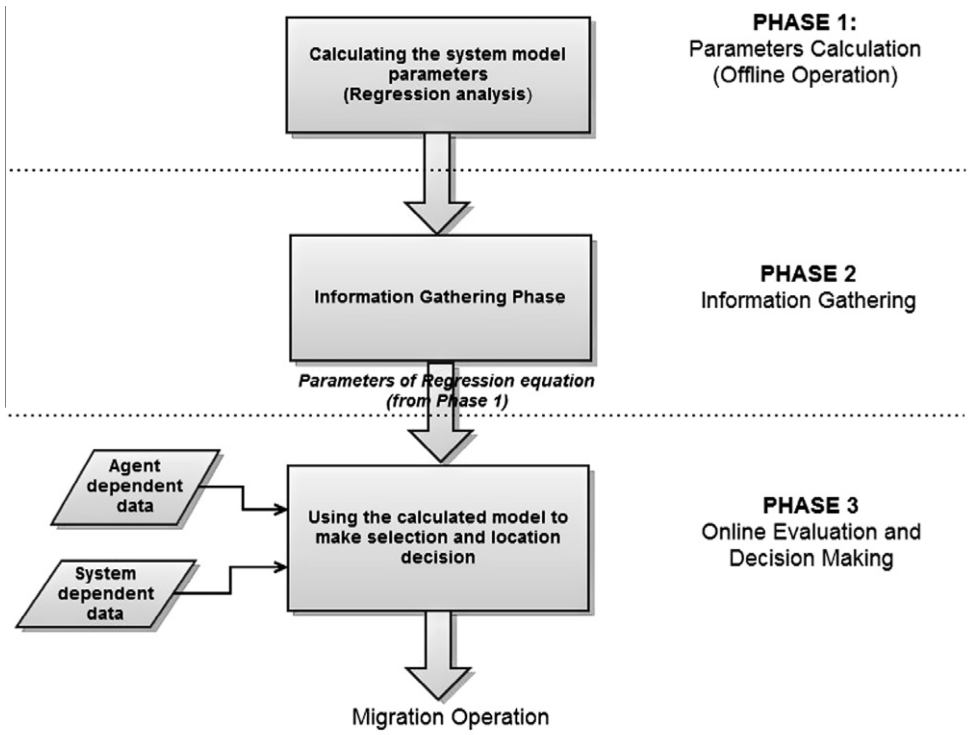

Calculating the system model parameters (Regression analysis)
PHASE 1:
Parameters Calculation (Offline Operation)
Information Gathering Phase
PHASE 2
Information Gathering
Parameters of Regression equation (from Phase 1)
Agent dependent data
System dependent data
Using the calculated model to make selection and location decision
PHASE 3
Online Evaluation and Decision Making
Migration Operation

Figure 1 Proposed system operation phases.

3.2.1. Information gathering policy

In the proposed system, we employed an event-driven information gathering policy. Specifically, the node that updates its local load value due to a local event will evaluate its Location Credit (LC) value, which represents the local load value, and send it to the other Load Balancing Coordinator LBC(s) agents in the system.

There are different types of local event as follows: (1) Local Agent Termination: a local Work Agent's life cycle is ended. (2) Local Agent Start: a new local Work Agent will be started. (3) Local Agent Migration: the local node has just executed a migration process and one of its local agents has been migrated. (4) Incoming Migrating Agent: the local node has been selected as destination for the migrating agent. (5) A Mobile Agent Departure: as a system of mobile agents, each agent has its own itinerary path to follow in order to accomplish its services, thus a work agent may leave the local node to complete its job at another node which is the next destination in its itinerary path. (6) A Mobile Agent Arrival: the local node is the destination in the agent's itinerary path.

In each of the above mentioned events the local load value is changed, whether by increase or decrease according to the situation. The load status is determined by comparing the Location Credit (LC) value to the Threshold value, the updated LC value is sent to the other LBC(s) so that they update their list of candidate nodes.

3.2.2. Selection policy

The selection policy deals with which task or agent is migrated whenever there is a need. We assign a numerical value, called "**The Aggregate Selection Criteria**" ASC, to every agent. The ASC value indicates the tendency of the agent to remain undisturbed in case migration is under consideration. To an agent, the higher its ASC value, the higher its chance to stay at the same machine. In other words, that its chance to be selected for migration is lower.

The ASC value of an agent is assumed to depend on two types of parameters, namely:

Agent dependent parameters:

- Its computational load, as it represents a main source of host loading.
- Its communication load, as it represents another source of host loading.
- Agent's size, an agent travels through network to its new destination, thus an agent with a big size is expected to take more time to reach its destination node.
- Agent's priority, the interruption of a high priority agent running to perform a migration process should be avoided.

System dependent parameters:

- Reliability of communication path between hosts, the migrating agent delivery is not guaranteed when the physical path reliability is low.
- Availability of needed resource on the source host, this factor represents the affinity between an agent and its running host.
- Source Host's loading, the host with high load will be more subject to let some agents migrate.

Other factors may be considered to study their effect on the agent selection decision, for example the host resource usage is not included in the ASC equation. Also agent mobility to the next host as part of the agent itinerary is not considered in the ASC equation, the mobile agent is assumed to know its next destination at the end of its service execution on the local node.

The ASC value of an agent increases in the following ways:

- Agent's workload decreases.
- It communicates frequently with local work agents.
- It has a high affinity with the local machine. For example, it requires a special type of processors, I/O devices, or large amounts of data localized at the machine.
- Agent's remaining execution time is short.
- Agent's size is large.
- Communication load is small.
- Communication path between hosts is not reliable.
- Needed resource is available.
- Agent has high priority.

Using a multiple linear regression operation, we will try to gather all the mentioned factors into one equation.

As shown in Table 1, the independent variables are quantitative or categorical. Because categorical predictor variables cannot be entered directly into a regression model and be meaningfully interpreted, some other method of dealing with information of this type must be developed. There are two commonly used methods for coding categorical variables so they can be used in regression models, *dummy coding* and *effect coding* [17]. To include the categorical variables in the equation, we will use the simplest way which is the dummy coding, where we assign a 1 for the analysis belonging to that category and 0 to the units not belonging to it. A categorical variable with k levels will be transformed into $k-1$ variables each with two levels. For example, if a categorical variable had three levels (three categorical values), then two dichotomous variables could be constructed that would contain the same information as the single categorical variable.

Table 2 summarizes the proposed effecting variables and their coding. Therefore the complete equation can be written as:

$$\begin{aligned}\mathrm{ASC}_i = {} & b_0 + b_1 W_i + b_2 U_i + b_3 R_i + b_4 Ld_{i1} + b_5 L_{di2} \\ & + b_6 H_{i1} + b_7 H_{i2} + b_8 P_{i1} + b_9 Pi_2 + b_{10} S_i \end{aligned} \tag{2}$$

Assuming that the expected value of the disturbance term is zero.

3.2.3. Location policy

The location policy determines to which destination machine the selected agent will be migrated to. The selection of the destination host is based on the **Location Credit** (LC) value calculated by the LBC agent, this value is used locally to determine

Table 1 Proposed effective variables on the ASC value.

Load components	Categorical variable
Agent's computational load	Host's loading, resource availability
Agent's communication load	Communication reliability
	Agent's size, agent's priority

Table 2 Regression variables.

Regression variable	Regression parameter, b_k	Description
Computational load, W_i	b_1	W_i (Agent's comp. load value)
Communication load, U_i	b_2	U_i (Agent's comm. load value)
Resource availability: R_i	b_3	R_i = 1 available
		0 not available
Host's loading:		Ld_{i1}, Ld_{i2} = 0, 0 high
Ld_{i1},	b_4	1, 0 moderate
Ld_{i2}	b_5	1, 1 low
Communication reliability:		H_{i1}, H_{i2} = 0, 0 high
H_{i1},	b_6	1, 0 moderate
H_{i2}	b_7	1, 1 low
Agent's priority	b_8	P_{i1}, P_{i2} = 0, 0 low
P_{i1},	b_9	1, 0 moderate
P_{i2}		1, 1 high
Agent's size	b_{10}	S_i = 0 small
S_i		1 medium

the nodal load and compare it to the threshold value, while it is used by the other nodes to make the destination selection decision.

The main and only factor for destination selection is its **local load**; in this section we will describe the factors affecting the local load.

The local host load is dependent on the agents load running on that host. The load of an agent executing on a machine is defined as the sum of its computational load and the communication load in time unit [1].

Thus a local host load can be modeled as follows:

$$L_k = (W_k + U_k) \tag{3}$$

where W_k is the total computational load value of running agents on machine k. U_k is the total communication load value of running agents on machine k. L_k is the total local load on machine k.

The previous relation is valid for a system of identical machines, i.e. they have identical specifications; any two hosts will suffer the same load if they run the same number of agents.

Another case for non-identical machines is proposed in [18], each machine v has a capacity cap_v. The load on v is the sum of the assigned tasks weights divided by the capacity.

$$L_k = (W_k + U_k)/cap_v \tag{4}$$

where W_k is the total computational load value of running agents on machine k. U_k is the total communication load value of running agents on machine k. cap_v represents a specific capacity or throughput. L_k is the total local load on machine k.

Since the sum weight of the computation load and the communication load may differ according to some agent and system parameters, we will perform a multiple linear regression analysis to get the coefficient of each variable.

Therefore the complete Location Credit equation can be written as:

$$LC_k = b_0 + b_1 W_k + b_2 U_k \tag{5}$$

where W_k is the sum of the computational load of all running agents on host k. U_k is the sum of the communication load of all running agents on host k.

3.3. Agent's types

The proposed load balancing system aims to take advantage from the agent characteristic to create an autonomous system. This system consists of seven types of agents, the type which does all the computation and resides at the compute hosts is **Work Agent**. The majority of agents belong to this type; there may be one or more Work Agents in each compute host. It is the only type of agent that is subject to migration. The second type is **the Load Balancing Coordinator (LBC) Agent** which resides at each compute host. It needs to startup the other agent types destined to live at the same compute host, the LBC Agent prepares the list of candidate agents and the list of candidate nodes and update them when needed. The LBC Agent works in cooperation with the Host Agent and the Submitter Agent to execute the migration process [16].

The third type is **the Host Agent** which is responsible for keeping track of the number of alive Work Agents on the local host. It is necessary to have one Host Agent on each node. The fourth type is **the Submitter Agent** which is responsible for the delivery of the migrating agent to the selected host. Each node has its own Submitter Agent that waits for an Acknowledgment from the migrated agent once it reaches its destination host. The submitter Agent makes sure that the migrating agent is successfully received and initiated at the destination node. All LBC agents are started up by the **Central Agent** which is the fifth agent type, there is only one central agent residing on a central host. During every system run, statistics about the workload variations and migration decisions are recorded. At each migration point, the workload distribution situation before and after migration provide useful indicators about whether the current policies can really offer good load-balancing judgment. The sixth type is **the Threshold Agent**; it should be running on each node, it is responsible for comparing the local load value with the predefined threshold value to define the local node state. The last type is **the Communication Agent** is responsible for the messages delivery between the different types of agents. Each agent is addressed by its own id.

3.4. Proposed system operation

A dedicated agent on each host is responsible for the local coordination of load balancing activity. Each such Load Balancing Coordinator (LBC) decides under which circumstances to initiate migrations as well as to accept incoming migrations.

Although LBCs are autonomous, they act as a team in order to establish the load balancing infrastructure. The nodal operation is described in the flowchart in Fig. 2; detailed description for each process will be followed:

3.4.1. Preparing the list of candidate agents

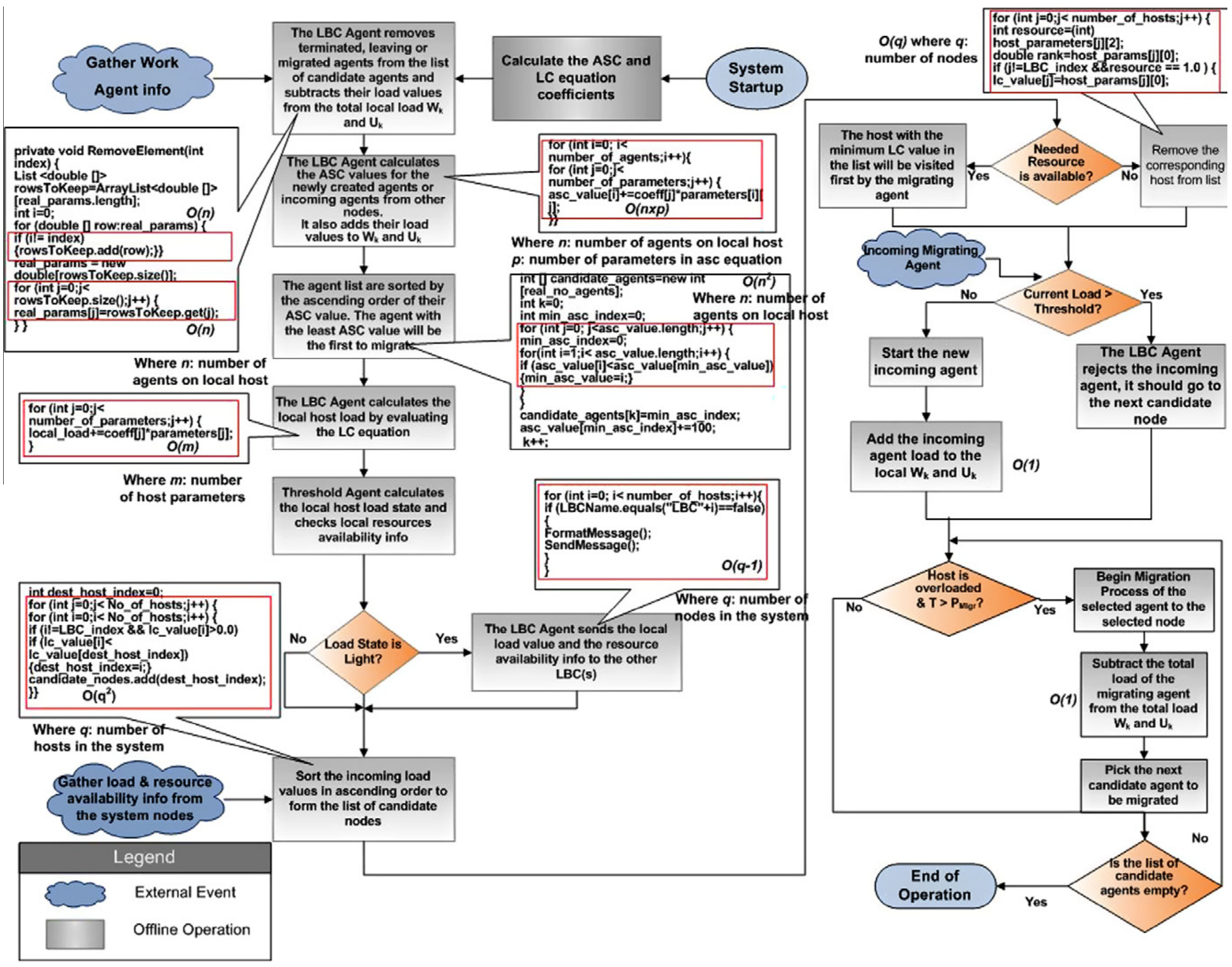
After gathering information about the local Work Agents alive on the local host, the LBC Agent prepares the list of candidate agents for migration. The LBC Agent should first delete the locally terminated and the migrated agents from the list of candidate agents. It also subtracts their corresponding computational and communication load values from the total local computation and communication values W_k and U_k as follows:

$$W_k = W'_k - W_i \quad (6)$$

$$U_k = U'_k - U_i \quad (7)$$

where W_k is the total computation load after migrating the agent or the local agent termination. U_k is the total communication load after migrating the agent or the local agent termination. W'_k is the total computation load before migrating the agent or the local agent termination. U'_k is the total communication load before migrating the agent or the local agent termination. W_i is the total computation load of the migrated or the terminated agent. U_i is the total communication load of the migrated or the terminated agent. Equally, the total load value for the newly created agents should also be added to the total local load.

The LBC Agent then evaluates the ASC equation for each work agent and then sorts them in the ascending order of their ASC values. The first agent to be migrated will be the agent with the least ASC value. Fig. 2 also shows the detailed implementation of each operation and its complexity analysis, one can deduce that removing an agent from the list has complexity $O(2n * v)$ where v is the number of deleted agents and n is the number of agents on the local host. The next operation is the ASC equation evaluation which is $O(n * p)$ where p is the number of variables in the ASC equation. Then finally is the sorting operation of the candidate node list which is $O(n^2)$. Then the overall complexity is:



Gather Work Agent info
The LBC Agent removes terminated, leaving or migrated agents from the list of candidate agents and subtracts their load values from the total local load W_k and U_k
Calculate the ASC and LC equation coefficients
System Startup
The LBC Agent calculates the ASC values for the newly created agents or incoming agents from other nodes. It also adds their load values to W_k and U_k
The agent list are sorted by the ascending order of their ASC value. The agent with the least ASC value will be the first to migrate
The LBC Agent calculates the local host load by evaluating the LC equation
Threshold Agent calculates the local host load state and checks local resources availability info
Load State is Light?
No
Yes
The LBC Agent sends the local load value and the resource availability info to the other LBC(s)
Sort the incoming load values in ascending order to form the list of candidate nodes
Gather load & resource availability info from the system nodes
Legend
External Event
Offline Operation
O(q) where q: number of nodes
The host with the minimum LC value in the list will be visited first by the migrating agent
Needed Resource is available?
Yes
No
Remove the corresponding host from list
Incoming Migrating Agent
Current Load > Threshold?
No
Yes
Start the new incoming agent
The LBC Agent rejects the incoming agent, it should go to the next candidate node
Add the incoming agent load to the local W_k and U_k
O(1)
Host is overloaded & T > P_{Migr}?
No
Yes
Begin Migration Process of the selected agent to the selected node
Subtract the total load of the migrating agent from the total load W_k and U_k
O(1)
Pick the next candidate agent to be migrated
Is the list of candidate agents empty?
No
Yes
End of Operation

```
private void RemoveElement(int
index) {
List <double []>
rowsToKeep=ArrayList<double []>
[real_params.length];
int i=0;                          O(n)
for (double [] row:real_params) {
if (i!= index)
{rowsToKeep.add(row);}}
real_params = new
double[rowsToKeep.size()];
for (int j=0;j<
rowsToKeep.size();j++) {
real_params[j]=rowsToKeep.get(j);
}}                                O(n)
```

Where n: number of agents on local host

```
for (int i=0; i<
number_of_agents;i++){
for (int j=0;j<
number_of_parameters;j++) {
asc_value[i]+=coeff[j]*parameters[i][
j];                O(nxp)
}}
```

Where n: number of agents on local host
p: number of parameters in asc equation

```
int [] candidate_agents=new int          O(n^2)
[real_no_agents];
int k=0;
int min_asc_index=0;
for (int j=0; j<asc_value.length;j++) {
min_asc_index=0;
for(int i=1;i< asc_value.length;i++) {
if (asc_value[i]<asc_value[min_asc_value])
{min_asc_value=i;}
}
}
candidate_agents[k]=min_asc_index;
asc_value[min_asc_index]+=100;
 k++;
```

Where n: number of agents on local host

```
for (int j=0;j<
number_of_parameters;j++) {
local_load+=coeff[j]*parameters[j];
}                          O(m)
```

Where m: number of host parameters

```
for (int i=0; i< number_of_hosts;i++){
if (LBCName.equals("LBC"+i)==false)
{
FormatMessage();
SendMessage();
}
}                                 O(q-1)
```

Where q: number of nodes in the system

```
int dest_host_index=0;
for (int j=0;j< No_of_hosts;j++) {
for (int i=0;i< No_of_hosts;i++) {
if (i!=LBC_index && lc_value[i]>0.0)
if (lc_value[i]<
lc_value[dest_host_index])
{dest_host_index=i;}
candidate_nodes.add(dest_host_index);
}}         O(q^2)
```

Where q: number of hosts in the system

```
for (int j=0;j< number_of_hosts;j++) {
int resource=(int)
host_parameters[j][2];
double rank=host_params[j][0];
if (j!=LBC_index &&resource == 1.0 ) {
lc_value[j]=host_params[j][0];
```

Figure 2 Nodal operation on local host.

$$O(2n * v) + O(n * p) + O(n^2) = O(n^2) \tag{8}$$

3.4.2. Local load calculation

After receiving the load information from the alive work agents on the local node, the LBC agent would be able to calculate the local load value by evaluating the LC equation.

Then the Threshold Agent will compare this value to the threshold value to determine the nodal load state. The Threshold Agent will send back the load state to the LBC Agent as well as the resource availability information.

If the local load state is light, the LBC Agent will send its local load value and the resource availability information to the other LBC(s), so that they add it to their list of candidate nodes for destination selection.

The detailed implementation is showed for the LC equation evaluation which is $O(m)$ and the sending process of the LC value to the other LBC(s) which is $O(q-1)$, where m is the number of host parameters and q: is the number of hosts in the system. Then the overall complexity is

$$O(m) + O(q-1) \tag{9}$$

3.4.3. Preparing the list of candidate nodes

When the LBC Agent receives the load information from the other LBC(s) in the system, it sorts them in the ascending order of their LC values. If the migrating agent requests a specific resource, the LBC Agent will exclude the candidate nodes that do not have this resource from the list.

From Fig. 2, one can deduce that the complexity of the candidate nodes list sorting is $O(q^2)$ and excluding the nodes with unavailable resources is $O(q)$. Thus the overall complexity is

$$O(q^2) + O(q) = O(q^2) \tag{10}$$

3.4.4. Incoming load acceptance and rejection at destination node

It has to be mentioned that the migrating agent travels to its new destination carrying the addresses of the candidate nodes, so that if, for any reason, the first candidate's load has increased and it refused to accept the incoming agent, the agent will not be lost. It will be directed to the next candidate.

If the migrating agent is accepted, the incoming agent total computation and communication load should be added to the total computation and communication load W_k and U_k respectively.

$$W_k = W'_k + W_i \tag{11}$$

$$U_k = U'_k + U_i \tag{12}$$

where W_k is the total computation load after receiving the migrating agent. U_k is the total communication load after receiving the migrating agent. W'_k is the total computation load before receiving the migrating agent. U'_k is the total communication load before receiving the migrating agent. W_i is the total computation load of the incoming agent. U_i is the total communication load of the incoming agent.

After a complexity analysis for the modules of each operation, the most time consuming operation is the candidate agent preparation and sorting:

$$O(n^2) + O(m) + O(q-1) + O(q^2) + O(1) + O(1) + O(1) = O(n^2) \tag{13}$$

where n is the number of agents running on one local node, q is number of nodes in the system, m is number of variables in the LC equation.

Since the number of agents would be larger than the number of nodes then the dominant order is $O(n^2)$.

Thus the algorithm is $O(n^2)$ where n is the number of agents running on one node.

3.5. System implementation

A real multi-agent distributed system is implemented using JADE platform. **JADE** (**J**ava **A**gent **DE**velopment Framework) is a software Framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications [19].

The agents communicate with each other using ACL (Agent Communication Language).

Here DF (Directory Facilitator) provides yellow page service to the agents, and AMS (Agent Management System) controls the access and usage of agent platform. Only one AMS exists in an agent platform. MTS (Message Transport Service) provides communication path between the agent platforms [20].

Mobile agent is a program that is capable of migrating autonomously in the heterogeneous distributed environment. The agent platform can be split on several hosts. Typically (but not necessarily) only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is a basic container of agents that provides a complete run time environment for agent execution and allows several agents to concurrently execute on the same host. The main-container is the container where the AMS and DF live. The other containers, instead, connect to the main container and provide a complete run-time environment for the execution of any set of JADE agents.

In order to startup the distributed system from a single point, WADE is used. **WADE** (Workflows and Agents Development Environment) is a domain independent software platform built on top of ***JADE*** [20] **.**

WADE inherits from JADE a distributed runtime composed of several "***Containers***" that can run on different hosts and can contain each one a number of ***agents***. Even if this is not strictly mandatory, most of the time a container corresponds to a JVM. The set of active containers is called a ***Platform***. As depicted in Fig. 3 a special container exists in the platform called "***Main Container***". The Main Container (hosting the JADE AMS and DF) must be activated first with other containers registering to it at bootstrap time.

During the parameters calculation phase, we will use the R tool to perform the regression analysis. **R** is a programming language and software environment for statistical computing and graphics. The R language has become a *de facto* standard among statisticians for the development of statistical software [21,22].

R is widely used for statistical software development and data analysis [22]. We will use R version 2.5.0.

R can perform multiple regression quite easily. The basic function is: lm(model, data), after providing the input independent variables values in form of CSV(Comma Separated Values) file. The coefficients are obtained in one step, by calling the lm function.

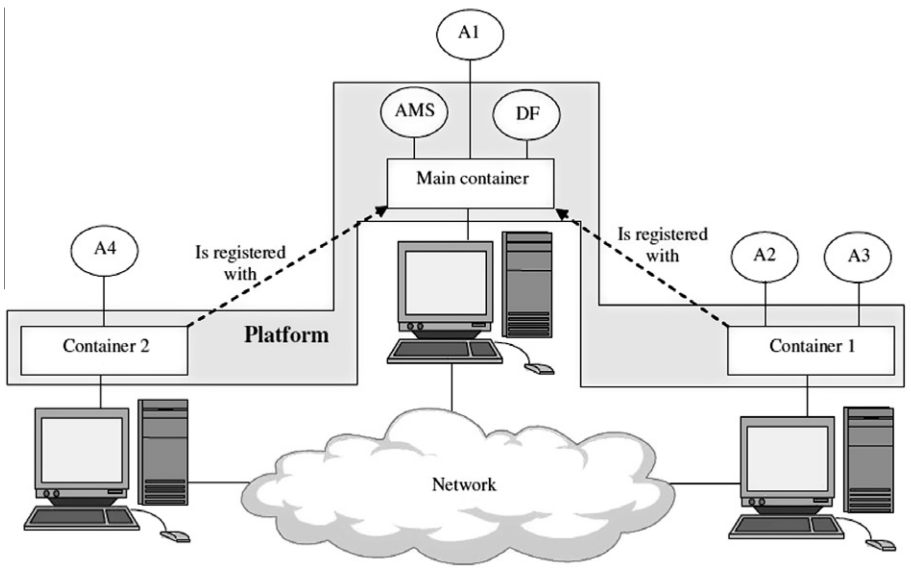


Figure 3 JADE distributed architecture.

4. Experiment environment

4.1. System overview

To investigate the effects of the proposed load balancing system, a multi-agent system will be implemented to provide a testing platform of the proposed load-balancing model. The whole system is written in Java using JDK 1.6 and is implemented on a cluster of 10 machines, running Linux 2.6.9 (Scientific Linux 4.5).

The configurations of these 10 machines are listed in Table 3. These machines reside at the High Performance & Grid Computing Laboratory of the Department of Computer and Systems, at the Electronics Research Institute in Dokki. All the machines have separate hard drives, but the same set of files is shared on every one of them. This is to ensure that all the related initialization and configuration files are accessible from every machine.

One host is the central host that holds the Main-Container, this host will invoke the whole system start up on each host.

4.2. Synthetic workload

The workloads fed as input to load balancing algorithm were obtained by creating artificial workloads with mathematical characteristics. This workload is executed by the Work Agents only.

Specifically, all the workloads were a random amount of program loops in which there are large amounts of numerical calculations. The execution time of this loop represents the computational part of the load value.

Table 3 Machines in the experimental system setup.

Model	Intel PC
Processor	3.40 GHz Intel Dual P4
OS	Linux 2.6.9
Main memory	1024 MB
Number of machines	10

The communication pattern is known in advance before execution, the time taken to send a message is the communication load.

The host load is calculated based on the obtained LC model in the offline phase, the host load state is obtained by comparing the LC value to the threshold value.

4.3. Experimental parameters

For each experiment, several parameters have to be specified. They are described in this section.

The dynamic model design is based on updating the load value at the corresponding host after each agent event (i.e. new agent start, migrating agent departure from source, migrating agent arrival at destination, etc.) and taking action based on the new load value as described in Fig. 2.

4.3.1. Number of hosts

The maximum number of nodes used in this experiment is 9 nodes. The nodes are connected through LAN. In this research, we are only concerned about homogenous machines for easier comparison of raw UNIX loading values from different hosts.

4.3.2. Number of agents

It is easily seen that the number of agents is one of the most important parameters in a multi-agent system experiment. We used the following set of agent numbers: (24, 48, 80 and 120 agents) as suggested in [1], although the machine specification and the OS version are different but it was preferred to stick to the same experiment parameters values to facilitate comparative analyses of results. When the average number of agents in a machine is relatively small, less choice is available for the selection policy and the effect of adding an agent to a machine or removing one is very significant, directly making the load-balancing algorithm less efficient. However, when the average number of agents in a machine becomes relatively large, the load balancing effect is largely seen.

Number of queries to study the load balancing effect from the user's point of view, it is necessary to study the agent's response time when applying load balancing and without applying it.

The running time is not constant for each experimental case, the test is stopped when all the nodes reach steady state, i.e. when the load is balanced on all nodes and no more agent migration takes place.

4.4. Performance metrics

One of the performance metrics is the execution time of a query which consists of a set of agent operations and interaction. The other is the workload distribution situation across all machines in a cluster. For the former proposed performance metric, some representative query to a system is needed. That means, if we adopt such a performance metric, a representative query must be devised in order that this metric can be obtained generically in any multi-agent systems. Such a query should specify what tasks agents will perform, how and with whom each of them will communicate, and when to do all these operations.

We implemented this by defining pairs of agents that will interact together to perform some computational work, the first agent sends a message to its peer requesting a certain information, when the other agent receives the message, it will make some computational work and send back the requested value, the time elapsed between sending the request and receiving the response is the value to be considered.

5. Results

5.1. Parameters calculation phase

In this section, we will give an overview about the regression analysis performed to calculate the coefficients of Eqs. (2) and (5) using the R tool. The input values fed into the R tool are random values which have a Guassian distribution. Each parameter has an average value which is based on a study of a real multi-agent system behavior.

5.1.1. The ASC equation

Table 4 shows the obtained coefficients from the regression analysis:

By substituting into Eq. (2):

$$ASC_i = 5.02362 - 1.22268W_i - 0.09419U_i + 1.09581R_i + 1.15273Ld_{i1} + 0.91098Ld_{i2} + 1.11610H_{i1} + 1.08454H_{i2} + 1.83428P_{i1} + 1.21858P_{i2} + 0.17135S_i \quad (14)$$

R^2 value = 98% which means that the model represents 98% of the input values.

The *P* value is order of zero which indicates a significant relationship between the dependent variable, ASC value, and at least one of the explanatory variables.

Having a big coefficient means that this variable will make the agent tends to stay rather than being migrated. Let us consider each coefficient and interpret its value meaning:

Computation load Coefficient: b_1 is relatively large negative value, which means that an agent having a big computation load is more likely to be migrated as its ASC value will be reduced.

Table 4 Obtained coefficients for the ASC model.

Variable	Coefficient	*P* value
X_1	−1.22268	6.56e−14***[a]
X_2	−0.09419	5.01e−11***
X_3	1.09581	1.06e−13***
X_4	1.15273	2.22e−12***
X_5	0.91098	1.87e−08***
X_6	1.11610	2.13e−10***
X_7	1.08454	5.67e−10***
X_8	1.83428	< 2e−16***
X_9	1.21858	3.06e−12***
X_{10}	0.17135	0.0867.*[b]

a: *** *P* value of order of 0.
b.: * *P* value of order of 0.1.

Communication load Coefficient: b_2 has a negative value, then we can deduce that an agent that communicates with an agent running on a distant node, i.e. has a big communication load, is more subject for migration as its ASC value will be small. Since b_2 has the smallest weight among the regression coefficients then it has the weakest effect on the ASC value and therefore the migrating agent selection.

Resource Availability Coefficient: b_3 has relatively large values because when $Ri = 1$ that's means that the agent finds the needed resource on the running host thus it is less subject for migration.

Host Load Coefficient: b_4 has higher weight than b_5 because when the running host is lightly or moderately loaded, it is less subject to select one of its agents to be migrated.

Reliability's Coefficient: b_6, b_7 have relatively large values because when $H_{i1}H_{i2} = 11$ that's means that the agent may not reach the destination node through the unreliable network, thus the migration frequency is less.

Priority's Coefficient: b_8 and b_9 have high weight because the high or moderate priority agent are less subject to be migrated.

Agent size Coefficient: b_{10} has a positive sign which means that a medium size agent will be less subject for migration as they will encounter more loads in transmission. Note that b_{10} has a relatively small weight among the regression coefficients. Then it has a weaker effect on the ASC value than the other variables.

5.1.2. The LC equation

Table 5 shows the obtained coefficients from the regression analysis:

By substituting the coefficients values into Eq. (5):

$$LC_k = 0.33125 + 1.06901W_k + 1.08380U_k \quad (15)$$

R^2 value = 96% which means that the model represents 96% of the input values. The P value is order of zero which indicates a significant relationship between the LC value and the local load.

Let us consider each coefficient and interpret its value meaning:

Computation load Coefficient: b_1 is a positive value, which means that a host that has a big computational load value will be excluded from the candidate destination nodes.

Communication load Coefficient: b_2 is a positive value, so when it is multiplied by the communication load, a host that has a big communication load value will be excluded from the candidate destination nodes.

Table 5 Obtained coefficients for the LC model.

Variable	Coefficient	*P* value
X_1	1.06901	4.36e−06***[c]
X_2	1.08380	0.000409***

[c] *** *P* value of order of 0.

5.2. Cluster load distribution

In this section, we will go through four experiments. Each one has different number of hosts and different number of agents as displayed in each plot.

By running each case, we will study the effect of load balancing on equally distributing the load over the cluster hosts. Fig. 4 shows that the overloaded host load is decreasing in each iteration till the system reaches steady state. Considering Fig. 4a, Host4 performs a migration operation at iteration 1 and one of its agents goes to Host 2. Thus the load is updated at source and destination as shown in Table 6.

5.3. Query response time

This experiment aims to quantify the performance of the proposed system from the user's point of view. The response time of different number of queries is studied with and without applying load balancing.

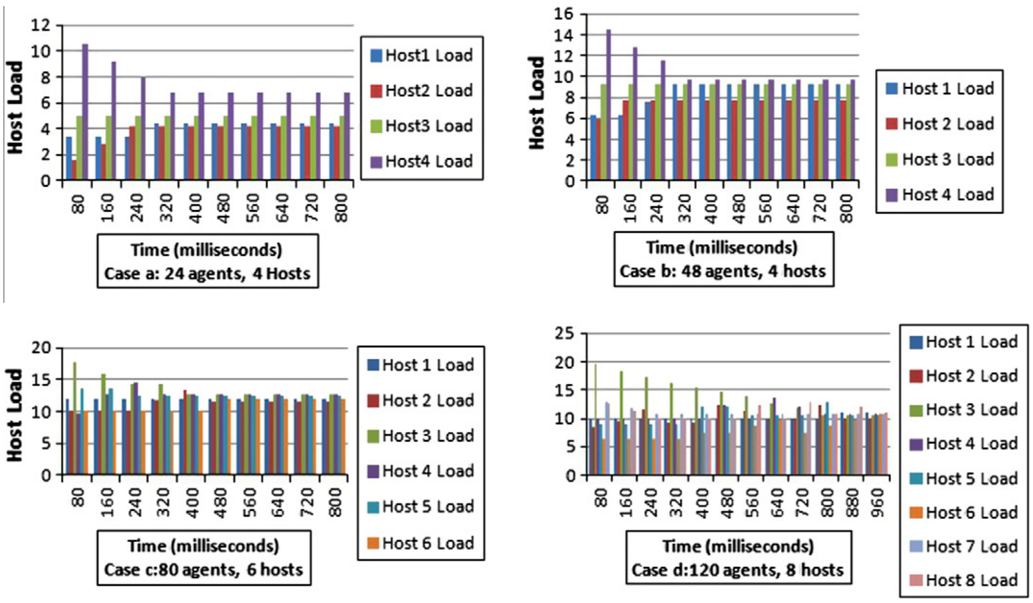


Figure 4 Cluster load distribution.

Table 6 Load update after iteration 1.

	Total computational load, W_k	Total communication load, U_k	LC value
Host 4 (source node)	7.194 – 1.079 = 6.115 Based on Eq. (6)	2.337 – 0.163 = 2.174 Based on Eq. (7)	9.224427 Based on Eq. (15)
Host 2 (destination node)	0.663 + 1.079 = 1.742 Based on Eq. (11)	0.471 + 0.163 = 0.634 Based on Eq. (12)	2.88059 Based on Eq. (15)

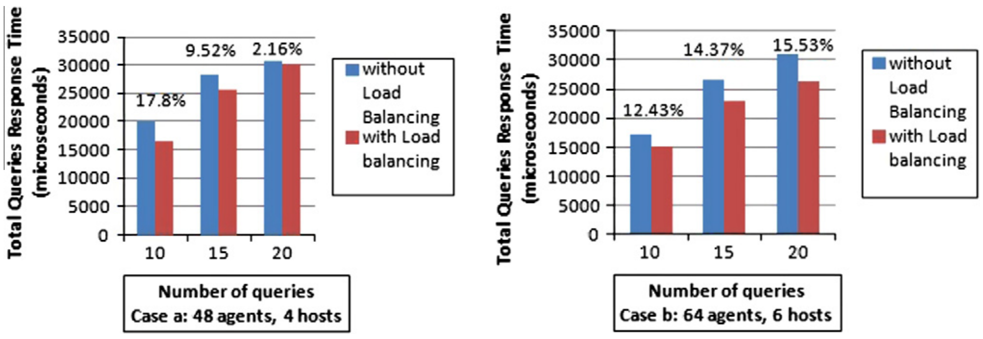


Figure 5 Query response time.

We used 2 test cases, 48 work agents running on 4 hosts and 64 work agents running on 6 hosts. Each case run 3 sets of queries 10, 15 and 20 as shown in Fig. 5.

The speedup in execution is due to the fair load distribution between the cluster nodes. The speedup is shown in figures for each case.

$$\text{Speed up }\% = (QRT_1 - QRT_2)/QRT_1 \quad (16)$$

where QRT_1 and QRT_2 are the Total Query Response Time without and with applying load balancing respectively.

By comparing the response time in Fig. 5a and b, we can see that the speedup is lower when applying load balancing to case b especially for the 20 queries case. One can deduce that the load balancing performance is related to the load index [23], i.e. when the number of tasks is higher than the hosts capacity, even when redistributing the agents over the cluster, all the hosts become highly loaded and this badly effects the response time.

6. System comparison

In this section we will examine the differences and similarities between our proposed system and three other previous load

Table 7 Comparison with recent load balancing schemes.

	Scheme 1 Our proposed system	Scheme 2 Comet Algorithm [1]	Scheme 3 Prediction based load balancing [14]	Scheme 4 Hybrid model [15]
System configuration	The system compromises a set of compute nodes decentralized in control, with seven different types of agents	The system compromises a set of compute nodes centralized in control, with three different types of agents	The system compromises a set of compute nodes centralized in control, with three different types of agents	The system compromises a set of compute nodes decentralized in control, with three different types of agents
Information gathering policy	Event-based information gathering	Periodic information gathering, the length of period can be changed by the system administrator	Periodic information gathering, the length of period is fixed	Periodic information gathering. The length of information gathering period is not defined
Agent selection policy	More factors are considered to calculate the credit value for agent selection	Credit-based selection depending only on the agent computational and communication load	Same criteria as scheme 2	Agent with the highest load is selected for migration. Agent's load is defined its computational and communication load
Location selection policy	The destination node is the node with the least LC (Location Credit) value	The destination node is the node that its agents exchange the biggest communication load with the migrating agent	Same criteria as scheme 2	Same criteria as scheme 2
Decision making centrality	Migration decision is taken locally by the LBC (Load Balancing Coordinator) Agent on each node	Migration decision is taken by one central Agent located on one central node	Migration decision is taken by one LBCA (Load Balancing Control Agent) located on only one node	Migration decision is taken locally by the LBC (Load Balancing Coordinator) Agent on each node
Migration condition	Local load value is greater than the load threshold value	Same as scheme 1	If the predicted value is larger than the current value, indicating further load increase till overload, then migration will take place from this node	Same as scheme 1
Complexity analysis	The overall operation is $O(n^2)$ where n is the number of agents running on one node	Not calculated	Not calculated	Not calculated
Applicability	An integrated nodal control system in MAS (Mobile Agent System) that may be part of the nodal middleware	The implemented model is integrated into a distributed Multi-agent System; the system is built on top of the JATLite platform	The implemented model is integrated to a MAS (Mobile Agent System) supporting Web service built on top of JADE	The implemented model is integrated into a Multi-agent System; the system is built on top of the JADE platform

balancing schemes; namely Comet Algorithm [1], Prediction-based dynamic load balancing [14] and Hybrid Load Balancing Model [15] as shown in Table 7.

7. Conclusion and future work

In this research, we have presented a dynamic load balancing scheme in a multi-agent system based on using linear regression equations to make the selection and location decision. This research also made use of the agents' characteristics to design a load balancing scheme for multi-agent applications for distributed systems. The proposed dynamic load balancing uses the credit-based criteria as in [1], however more factors are considered to calculate the credit value for both agent and host selections, the effect of those factors are studied through a multiple linear regression analysis. We do not claim that the given model is a 'complete' model, encompassing ALL important parameters and neglecting NON-IMPORTANT ones. It is rather an attempt to investigate the possibility of using regression in studying such a multi-sided problem. It should be stated that the model is rather but not overly simplified, in terms of its linearity and in the number of variables included in the model.

Yet, we have tried as much as possible to include parameters that have been judged by most researchers as 'important'. Some 'non-mentioned' parameters may be implicitly defined within some other 'explicitly mentioned' parameters. For example, agent-host affinity can be implied in the resource availability. Also, it is assumed that all agents are 'mobile' with respect to any node (i.e. without restrictions).

A test multi-agent platform is built, different sets of system parameters are fed into the system and results were collected. The results show the effect of the proposed system on the cluster load distribution, the cluster nodes load values are nearly equalized under the threshold value. This effect of load balancing is also seen from the user side, by reducing the query response time.

Agents in the proposed model are assumed to move either in case of 'overload' condition on system nodes or to complete its itinerary path. The generality of the model stems from its applicability on 'any loaded' source node and 'any under-loaded' destination node. Moreover, it is platform independent and can be part of the middleware running on system nodes.

The limitations mentioned about the implemented system are:

- Firstly, the reliability problem, there is no guarantee that the migrating agent will reach the destination host. The source node does not keep a copy of the agent before it is left to its new destination. Further solutions must be found to provide more reliability for the migrating agent.
- The second limitation, the time taken to execute a migration process is not explicitly calculated. A study [24] shows that the JADE platform executes one migration operation in nearly 30 microseconds in a homogenous cluster, thus we assumed that it will be negligible.
- The agent mobility to the next host as part of the agent itinerary is not considered in the ASC equation, the mobile agent may be selected for migration before it completes its service on the local host.
- Another limitation is the assumption that the communication pattern is known before the system startup, which is not always the case. It also remains static during the system operation. In reality, agents may change their communication patterns dynamically. Thus, a further research direction is to investigate the effectiveness of the proposed system under such a more dynamic situation.
- Although the experimental work is performed on a homogenous cluster machines, the LC equation is formulated to cope with heterogeneity of the system machines. The coefficient of the computation load and communication load will vary according to the used machine capability.

References

[1] Chow K-P, Kwok Y-K. On load balancing for distributed multiagent computing. IEEE Trans Parallel Distrib Syst 2002;13(8):1153–61.
[2] Griss ML, Pour G. Accelerating development with agent components. Computer 2001;34(5):37–43.
[3] Luo Y, Davis DN, Lui K. A multi-agent system framework for decision support in Stock Trading. IEEE Netw Mag Spec Issue Enterp Netw Serv 2002;16(1).
[4] Papazoglou MP. Agent-oriented technology in support of EBusiness. Comm ACM 2001;44(4):71–8.
[5] Su Ch, Wu Ch. JADE implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring. Appl Soft Comput 2011;11:315–25.
[6] Kim YH, Han S, Lyu CH, Youn HY. An efficient dynamic load balancing scheme for multi-agent system reflecting agent workload. In: The 12th IEEE international conference on computational science and, engineering; 2009.
[7] DeGroot MH. Optimal statistical decisions. Newyork: McGraw-Hill, Inc.; 1970.
[8] Stender J, Kaiser S, Albayrak S. Mobility-based runtime load balancing in multi-agent systems, (SEKE'06), Reedwood City, CA, USA; 2006.
[9] Jiang YC, Li ZF. Locality-sensitive task allocation and load balancing in networked multiagent systems: talent versus centrality. J Parallel Distrib Comput 2011;71(6):822–36.
[10] Jiang Y, Jiang J. Contextual resource negotiation-based task allocation and load balancing in complex software systems. IEEE Trans Parallel Distrib Syst 2009;20(5):641–53.
[11] Cho Cho Myint, Khin Mar Lar Tun, A Framework of Using Mobile Agent to Achieve Efficient Load Balancing in Cluster. In: Proc. 6th Asia Pacific symposium on information and telecommunication technologies; 2005.
[12] Cao J, Wang X, Das SK, A framework of using cooperating mobile agents to achieve load sharing in distributed web server groups. In: Proceedings of the fifth international conference on algorithms and architectures for parallel processing (ICA3PP.02); 2002.
[13] Kumar Sinha P, Dhore SR. Multi-agent optimized load balancing using spanning tree for mobile services. J Comput Appl 2010;1(6):33–40.
[14] Son BH, Lee SW, Youn HY. Prediction-based dynamic load balancing using agent migration for multi-agent system. In: Proc. HPCC; 2010. p. 485–90.
[15] Ozcan I, Bora S. A hybrid load balancing model for multi-agent systems. In: Innovation in intelligent systems and applications (INSTA); 2011. p. 182–7.
[16] Nehra N, Patel RB, Bhat VK. A framework for distributed dynamic load balancing in heterogeneous cluster. J Comput Sci 2007;3(1):14–24.

[17] Gupta R. Coding categorical variables in regression models: dummy and effect coding. Cornell Statistical Consulting Unit; 2008, May.

[18] Westbrook J. On the power of preemption. Yale University, New Haven: Department of Computer Science; 1994, January.

[19] FIPA – Foundation for Intelligent Physical Agents. <http://www.fipa.org/>.

[20] G. Caire, WADE USER GUIDE; 2010.

[21] Fox John, Andersen Robert, Using the R statistical computing environment to teach social statistics courses. Department of Sociology, McMaster University; 2005.

[22] Vance Ashlee. Data Analysts Captivated by R's Power. New York Times; 2009.

[23] Zhou S. Performance studies of dynamic load balancing in distributed systems. Berkeley: University of California; 1987.

[24] Chmiel K, Tomiak D, Gawinecki M, Karczmarek P, Szymczak M, Paprzycki M. Testing the Efficiency of JADE Agent Platform, ispdc. In: Third international symposium on parallel and distributed computing/third international workshop on algorithms, models and tools for parallel computing on heterogeneous networks (ISPDC/HeteroPar'04),Cork, Ireland; 2004. p. 49–56.

Eng. Maha A. Metawei is currently a research assistant at the Computers and Systems Department, member of the High Performance & Grid Computing Group, Electronics Research Institute.

Prof. Salma A. Ghoneim is currently a professor at the Computers and Systems Department., Faculty of Engineering, Ain Shams University.

Dr. Sahar M. Haggag is currently an assistant professor at the Computers and Systems Department, Faculty of Engineering, Ain Shams University.

Prof. Salwa M. Nassar is currently the acting president of the Electronics Research Institute, and Head of High Performance & Grid Computing Group.