Chapter

Agent Based Load Balancing in Grid Computing

Wided Ali and Fatima Bouakkaz

Abstract

Load-Balancing is an important problem in distributed heterogeneous systems. In this paper, an Agent-based load-balancing model is developed for implementation in a grid environment. Load balancing is realized via migration of worker agents from overloaded resources to underloaded ones. The proposed model purposes to take benefit of the multi-agent system characteristics to create an autonomous system. The Agent-based load balancing model is implemented using JADE (Java Agent Development Framework) and Alea 2 as a grid simulator. The use of MAS is discussed, concerning the solutions adopted for gathering information policy, location policy, selection policy, worker agents migration, and load balancing.

Keywords: load balancing, grid computing, agent migration, resource utilization, mobile agents

1. Introduction

Grid computing has appeared as an encouraging smart computing paradigm. Grid computing purposes to collect the power of geographically distributed heterogeneous, multiple-domain computational resources to offer high performance. To realize the encouraging potentials of grid computing, effective job scheduling, and load balancing algorithms are important. Such algorithms should be very scalable since these systems typically have thousands to millions of resources. They should also be flexible and be adaptive to task requirements.

The load balancing prevents the state in which some resources become overloaded while the others are underloaded or idle. Therefore, the use of a load balancing mechanism is expected to enhance reliability. The problem that can increase in this mechanism is related to the characteristics of the grid, which are resource variations, resource heterogeneity, application variety, and the dynamicity of grid environments. Multi-agent systems give encouraging features for resource managers. The scalability, reactivity, cooperation, proactivity, flexibility, autonomy, and robustness that characterize multi-agents system can help in the complex task of resource management in dynamic and changing environments.

Multi-agent distributed systems give an exciting solution to grid load balancing. An agent-based structure is developed to offer services for high performance-programming environments and applications that can be used on the grid computing environment. Software agents improve expandability, permitting the number of resources involved to rise easily, by providing services that include job scheduling, monitoring, and supervisory for the system.

Asynchronous communication, parallel actions, and autonomous operations of agents allow MAS to adjust to dynamic modifications of the grid environment, thereby enhancing the stability, fault tolerance, responsiveness, and reliability of the grid. Identifying key reasons to prove the convergence of MAS and grid is not an easy task. In this chapter, a new Agent-Based load balancing Model is presented. A hierarchical architecture with coordination is designed to ensure scalability and efficiency. Also, a multi-agent approach is applied to improve adaptability. Multi-agent system is implemented with the JADE (Java Agent Development) framework for grid load balancing. The chapter also discusses the difficulties and advantages surrounding the task of integrating multi-agent systems into grid computing.

The structure of the chapter is organized as follows: Section 2 describes Agent based load balancing architecture and implementation, Section 3 shows implemented algorithms. Finally, Sections 4 concludes the chapter with comments and discussion about current and future works.

2. Proposed agent based load balancing model

The proposed model is an extension of our previous works related to load balancing system [1, 2], which is integrated into our agent-based load balancing in a grid computing project.

In this section, we will introduce an Agent-based load balancing Model (ABLBM). We will mention the components of the system and the interaction between agents briefly. We will describe the new features we added to Agent- based load balancing Model in grid, UML Classes Diagram, UML Sequence Diagram, algorithms, and load balancing mechanism in detail.

2.1 The ABLBM framework

The UML Classes Diagram of the proposed model comprises fourteen connected classes as follows (**Figures 1** and **2**):

1. The Model class is linked by an aggregation relationship to the Level 2, Level 1, and Level 0 classes

2. The Level 0 class contains one and only one Grid Agent

3. The Level 1 class contains one or more cluster Agents

4. The Level 2 class contains one or more clusters and the cluster class contains one or more resources

5. A Resource can be associated with one or more users, and each user can submit one or more tasks.

6. The Resource class is linked by an aggregation relation to the cluster class

7. Grid Agent class can create one or more Cluster Agents

8. Cluster Agent can create one Migration Agent, one or more Resource Agents, LBC Agents, and Worker Agents.

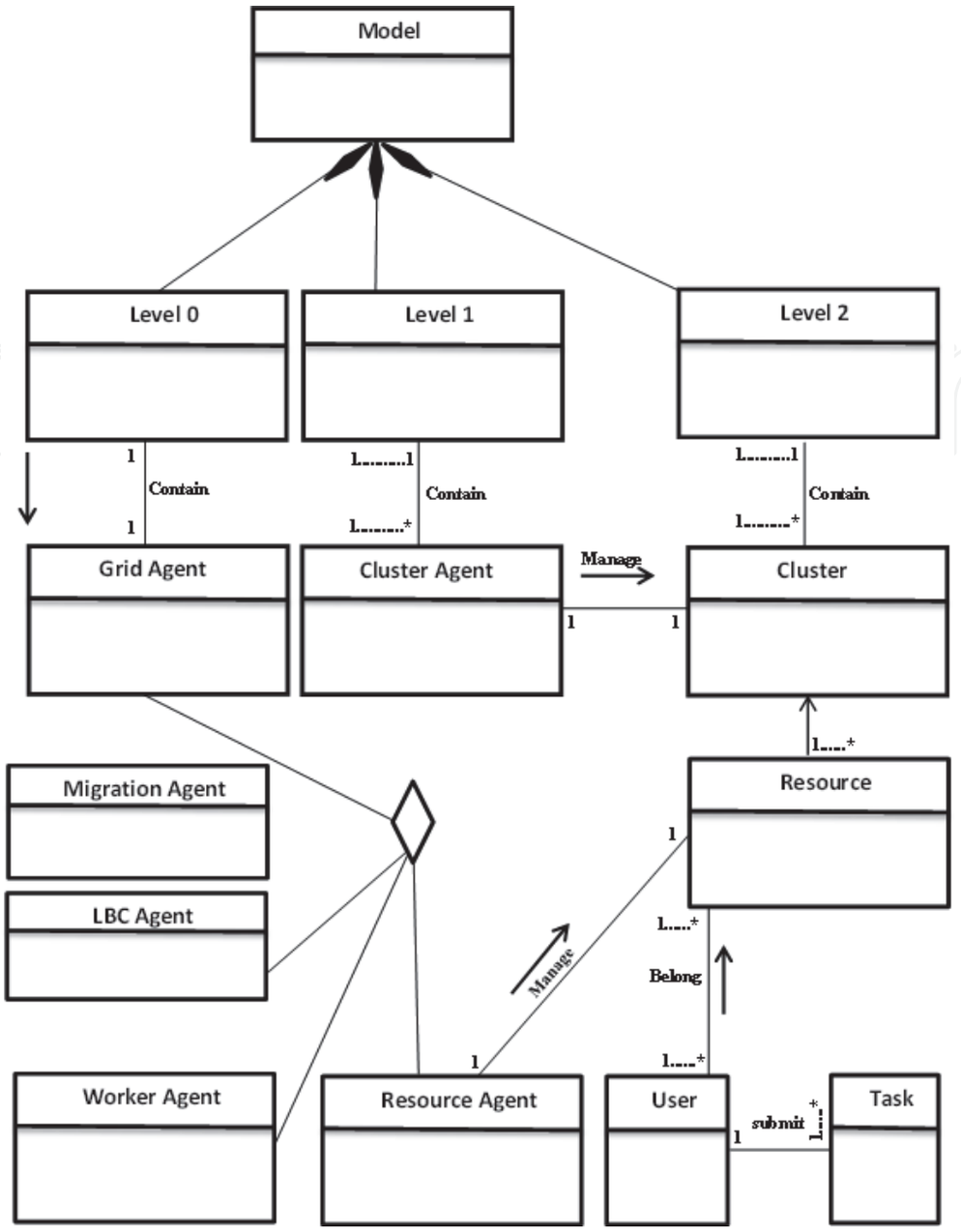

Figure 1.
UML classes diagram of ABLBM framework.

The proposed framework is intended to take advantage of the agent's characteristics to create a self-adaptive and self-sustaining load balancing system. The proposed system consists of six types of agents, in unbalanced situations, and if the Cluster Agent finds that there is a load imbalance between the resources under its control, it uses the Knowledge Algorithm to receive the load information from each Resource Agent. Based on this information and the estimated equilibrium threshold, it analyses the current load of the cluster. Depending on the outcome of this analysis, it decides whether to start a local balancing in case of an unbalanced state, or simply inform other Cluster Agent of its current load. Resource Agent sends the updated local load value to Cluster Agent, which updates its load information. Migration Agent is responsible for migrating Worker Agents to the selected underloaded resource. There is a Migration Agent in each Cluster, who expects acknowledgement of receipt from the receiving resource once it receives the migrated Worker Agent. The last agent is Grid Agent, it is the role of the distribution of jobs between clusters, and all Cluster Agents are started by this type of agents.

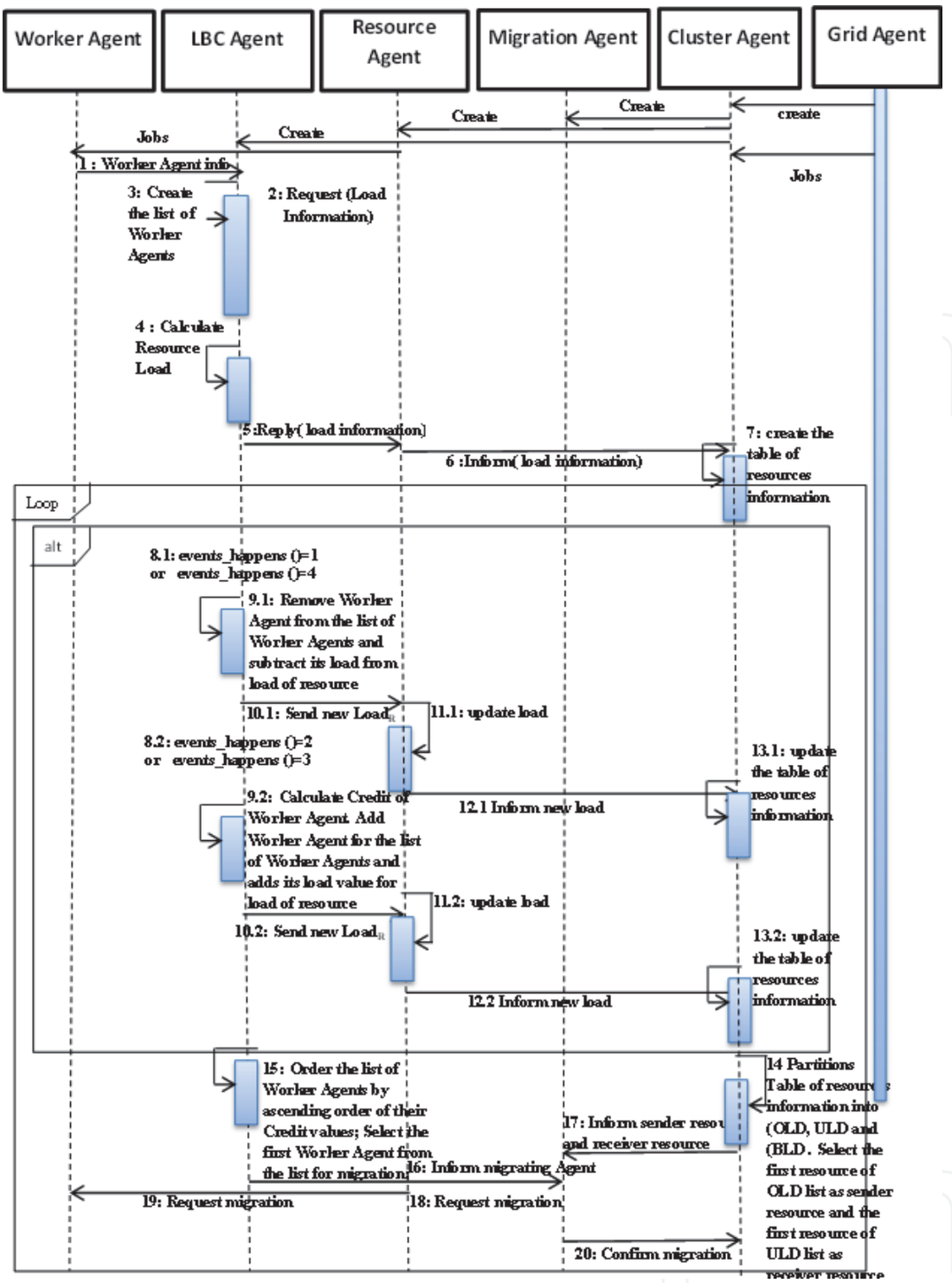

Figure 2.
UML sequence diagram describes agent interactions in intra cluster load balancing process framework.

2.2 Algorithms

We define two levels of load balancing algorithms: Intra-cluster load balancing and intra-Grid load balancing algorithm.

2.2.1 Intra cluster load balancing algorithm

This load balancing algorithm makes the imbalance situations can be resolved within a cluster. It is triggered when any Agent Cluster finds that there is a load imbalance between the resources which are under its control. To do this, the Agent

Cluster receives load information from each Resource Agent. Based on this information and the estimated balance threshold, it analyzes the current load of the cluster. According to the consequence of this analysis, it chooses whether to start a local balancing in the situation of imbalance state, or eventually just to notify other Agent Clusters about its current load. To implement this local load balancing, we propose the following three policies: load information gathering, agent selection policy, location policy, and Worker Agent migration.

2.2.1.1 Load information gathering

In the proposed algorithm, Agent Cluster decides to start a local balancing in the case of imbalance. There are some particular events that change the load configuration in a grid environment. The events can be categorized as follows:

1. Arrival of any new resource

2. Withdrawal of any existing resource.

3. Local Agent Worker Termination: a local Agent Worker's life cycle is ended

4. Local Agent Worker Start: a new local Agent Worker will be started

5. Incoming Migrating Agent: the local resource has been selected as a receiver for the migrating agent.

6. A Mobile Agent Departure: the local resource has been selected as a sender for the migrating agent.

7. Agent Worker ends the computation assigned to it, and becomes idle;

Whenever any of these activities happen the local load value is changed, Resource Agent sends the updated local load value to Agent Cluster who updates its Table of resources. Each Cluster Agent estimates its associated cluster capability by performing the following actions:

1. it estimates the current load of the cluster based on load information received from its Resource Agents;

2. it sends its load information to other Cluster Agents

The local host load is dependent on the Agent workers running on that host. A load of an agent executing on a machine is defined as the sum of its computational load and communication load in time unit [3].

A local host load can be defined as follows: The load L_k of a machine M_k is defined as the sum of its entire agents load on the host. More specifically

$$L_k = \sum_{M(ai)=k} (W_k + U_k) \tag{1}$$

A load of an agent executing on a machine is defined as the sum of its computational load and communication load $Load_{ij} = W_{ij} + U_{ij}$ Where: W_{ij} is Computational Load and U_{ij} is communication load.

2.2.1.2 Agent selection policy

The selection policy handles which Worker Agent is migrated whenever there is a necessity. We assign a numerical value, called credit, to every Worker Agent. The credit value designates the capacity of the agent to remain undisturbed in case of migration. For a Worker Agent, the higher its credit value, the higher its opportunity to stay at the same machine. In other words, its opportunity to be selected for migration is lower. The LBC Agent assigns credit to each Worker Agent and chooses which agent requests to be migrated using the credit. Any Worker Agent with high credit will be given more opportunity to preserve its current location (the resource the agent resides in) with less opportunity to be selected for migration.

The credit value of a Worker Agent is assumed to depend on two types of parameters, namely Worker Agent dependent parameters and System dependent parameters [4]:

2.2.1.3 Worker agent dependent parameters

1. Its computational load, as it represents the main source of resource loading.

2. Its communication load, as it represents another source of resource loading.

3. Agent's size, an agent migrates through the network to its new destination, thus an agent with big size is predictable to take more time to reach its destination resource.

4. Agent's priority, the interruption of a high priority agent running to perform a migration process should be prohibited.

2.2.1.4 System parameters

1. Reliability of the communication path between resources, the migrating agent delivery is not assured when the physical path reliability is low.

2. Availability of required resource on the source host, this factor represents the affinity between an agent and its running host.

3. Source Host's loading, the host with high load will be more subject to let some agents migrate.

The credit of a Worker Agent increases if the following situations [4]:

1. The Worker Agent's load reduces.

2. It communicates frequently with other worker Agents in other resources.

3. It has a high familiarity with the local machine. For example, it needs a special type of processors, I/O devices, or large volumes of data localized at the machine.

4. The agent's remaining execution time is short.

5. The agent's size is large.

6. The communication load is small.

7. The communication path between hosts is not reliable.

8. The needed Resource is available.

9. The agent has high priority.

In contrast, the credit value of Worker Agent reduces in the following situations:

1. The Worker Agent's load Increases

2. The communication with Worker Agents in other resources is increased.

3. Strong mobility or instant exchanges of messages (frequent message exchanges rises the Worker Agent's load.)

Using a multiple linear regression operation, we will try to gather all the mentioned factors into one equation, In linear regression, the relationship between a dependent variable, Y, and an independent variable X, is modeled by $Y = a + \beta X$. This interpretation of coefficient, it is appropriate only when the independent variable is continuous (quantitative). To incorporate qualitative independent variables into the regression model and formulate the model so the variables have interpretable coefficients, There are two commonly used methods for coding qualitative variables so they can be used in regression models, dummy coding and effect coding [4, 5]. To include the variables qualitative in the equation, we will use the simplest way which is the dummy coding, which assigns values "1" and "0" to reflect the presence and absence, For example, if we take a qualitative variable Resource availability $R_{i,}$ we assigned value 1 when the resource is available and 0 if it is not available. The final equation can be written as:

$$\begin{aligned}\text{Credit } A_i = {} & b_0 + b_1 W_i + b_2 U_i + b_3 R_i + b_4 Ld_{i1} + b_5 Ld_{i2} + b_6 H_{i1} + b_7 H_{i2} \\ & + b_8 P_{i1} + b_9 P_{i2} + b_{10} S_i\end{aligned} \tag{2}$$

Having a big coefficient means that this variable will make the agent tends to stay rather than being migrated.

b1: Computation load Coefficient: if b_1 is a relatively large negative value then an agent having a big computation load is more likely to be migrated as its credit value will be reduced. If b1 is a positive value then the resource that has a big computational load value will be excepted from the list of receiver resources.

b_2: Communication load Coefficient: if b_2 has a negative value, then we can assume that an agent has a big communication load, it is more matter to migration as its credit value will be small. Since b_2 has the smallest weight among the regression coefficients then it has the weakest effect on the credit value and therefore the migrating agent selection. If b_2 is a positive value, so when it is multiplied by the communication load, a resource that has a big communication load value will be excepted from the list of receiver resources.

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

b_4, b_5: Host Load Coefficient: b_4 has a higher load than b_5 because when the running host is underloaded or balanced, it is less matter to select one of its agents to be migrated.

b_6, b_7: Reliability's Coefficient: if H_{i1},H_{i2} = 1,1 then b_6, b_7 have relatively large values because that's means that the agent may not reach the destination node through the unreliable network, thus the migration frequency is less.

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

b_{10}: Agent size Coefficient: a big size agent mean b_{10} has a positive sign which means that a medium-size agent will be less matter for migration as they will encounter more loads in the transmission.

2.2.1.5 Location policy

After a Worker Agent is determined to migrate, we have to select the receiving resource. The location policy defines to which destination resource the selected Worker Agent will be migrated to. The Cluster Agent selects the destination resource. For this purpose, it executes the following actions:

When the Cluster Agent receives the load information from its resources, it Partition cluster into an overloaded resources list (OLD), under-loaded resources list (ULD), and balanced list (BLD) and it sorts OLD by descending order of their load and ULD by ascending order of their load. The resource will be in an overloaded list if its load is high. The resource will be in the underloaded list if its load is low. The resource is not into the overloaded list or the underloaded, after that the Cluster agent sorts the overloaded resources list by descending order relative to their Load and sorts underloaded resources list by ascending order relative to Their Load. In the next step, Cluster Agent determines the sender resource and the receiver resource, where the sender resource is the first resource in the overloaded resources list and the receiver is the first resource in the underloaded resources list. Each Worker agent records the communication load between all the resources. If the receiver resource has the highest communication load with the migrated Agent then it is selected as the receiver resource else we must select another receiver resource from the list of underloaded resources. This is because, if a receiver resource is an external resource, the load of Worker Agent may not reduce due to large external communication. Instead, the load may rise.

2.2.1.6 Worker agent migration

The Worker Agent selection is related to its credit value while the receiver resource is the most under loaded resource. The migration decision is taken by a Cluster Agent that sends it for Migration Agent associated. The proposed MAS employs a mobile-agent system to support the migration of an agent. For migrating the Worker Agents, the status of the system and the agents currently operate or registered have to be considered. The receiver resource has to have more than one running agent. The Migration Agent sends a request message to the AMS agent. Then, the AMS sends an authentication message along with timestamp to it. The Migration Agent sends a request message of migration along with the authentication message to the DLA (Dynamic Library Agent) of the receiver resource. The DLA then sends the Worker Agent code after verifying the authentication and validation of the message. Finally, the Worker Agent migrates itself to the receiver resource or migrates a clone agent, in calling the doMove () method by the migrating agent with as parameter the receiver resource. The migrated agent is executed by the Dynamic Library Agent, and if the migrated one is a clone agent, it records itself to the platform by itself.

```
LBC Agent: Gathering Information Algorithm
Input: Worker Agents info
Output: Load_R = resource load
{
Creates the list of Worker Agents;
Calculates the total local load host Load_R by using Eq. (1) and sends it for Resource Agent
Send Load_R to its Resource Agent associated
Loop wait for load change//depends on happening of any of defined events
{
if (events_happens () = 2 or events_happens () = 3) then
        {
        Calculates Credits of Worker Agents (by using Eq. (2);
        Adds Worker Agents for the list of Worker Agents and adds their load
values for the load of resource.
Sends Load_R to its Resource Agent associated;
        }
if (events_happens () = 1 or events_happens () = 4) then
        {
        Removes Worker Agents from the list of Worker Agents and subtract their
load from the load of resource
Sends Load_R to its Resource Agent associated;
        }
End Loop
}
```

```
Function events_happens ()
output Type: integer
If (Worker Agent Termination) then events_happens () =1; End If
If (Worker Agent Start) then events_happens () =2; End If
If (Incoming Migrating Worker Agent) then events_happens () = 3; End If
If (Worker Agent Departure) then events_happens () = 4; End If
If (Arrival of any new resource) then events_happens () = 5; End If
If (Withdrawal of any existing resource in the local host) then events_happens () = 6;End If
If (Load_cluster > S_threshold)then events_happens () = 7; End If//S_threshold saturation threshold
If (Cluster.state = unbalanced) then events_happens () = 8; End If
```

```
Resource Agent: Workload Estimation
Input: receive Load_R from LBC Agent, Worker Agents info
Output:
{
Started up LBC Agent associated;
Started up Worker Agents associated;
Receives Load_R from LBC Agent associated;
Sends Load_R to its Cluster Agent associated;
Keeping track of the number of alive Worker Agents on the local host;
}
```

Cluster Agent: Knowledge Algorithm
Input: receives tasks from AgentGrille, $Load_R$, $Load_C$
Output: Cluster Load, table of resources information
{
sends tasks among Resource Agents;
create the table of resources information;
receive $Load_R$ from the resource Agents under its control;
Updates the table of resources information;
if (events_happens () = 5) then
{
Create Resource Agent for the new resource;
Creates LBC Agent for the new resource;
Creates Worker Agents for new resource;
Sends tasks among Resource Agent of new resource
Adds information of the new resource for table of resources information
Updates the table of resources information;
}
if (events_happens () = 6) then
{
Kill Resource Agent of the destroyed resource;
Kills LBC Agent of the destroyed resource;
Kills Worker Agents for the destroyed resource;
Removes information of the destroyed resource from table of resources information
Updates the table of resources information;
}
Diffuses Cluster Load to other Cluster Agents;}

AgentLBC: Selection policy Algorithm
Input: AgentWorkers info
Output: AgentWorkers list are sorted by the ascending order of their credit value, selected AgentWorker
{
Orders the list of Worker Agents by ascending order of their Credit values;
Selects the first Worker Agent from the list for migration;
Sends this information for Migration Agent;
}

Cluster Agent: location policy algorithm
Input: table of resources information, $Load_{R,}Load_{cluster}$
Output: Sender Resource, Receiver Resource
{
if (events_happens () = 7)//cluster is saturated
intra-grid load balancing algorithm
Else
If (events_happens () =8) then
{
Partitions Table of resources information into overloaded resources table (OLD), under-loaded resources table (ULD) and balanced resources table (BLD)
OLD ← φ; ULD ← φ; BLD ← φ

For every $resource_i$ of cluster do
{
If ($resource_i$ is saturated) then OLD ← OLD ∪ $resource_i$;
Else Switch
Case 1:
$Load_R > B_{threshold}$: OLD ← OLD ∪ $resource_i$; /* $B_{threshold}$ is balanced threshold */
Case 2:
$Load_R < B_{threshold}$: ULD ← ULD ∪ $resource_i$;
Case3:
$Load_R = B_{threshold}$: BLD ← BLD∪ $resource_i$;
Sort OLD by descending order relative to their $Load_R$;
Sort ULD by ascending order relative to their $Load_R$;
Selects the first resource of OLD list as sender resource;
Selects the first resource of ULD list as receiver resource;
Sends this information for Migration Agent;

Migration Agent: Worker Agent Migration algorithm
Input: Sender Resource, Receiver Resource
Output: an Acknowledgment
Receives Sender Resource and Receiver Resource from its related Cluster Agent.
Sends migration request for AMS agent;
If receives an authentication message from AMS agent then
{
Sends a request message for Dynamic Library Agent of the receiver resource;
DL Agent sends code for the Worker Agent;
Worker Agent migrates itself to the receiver resource;
Waits for an Acknowledgment from the Dynamic Library Agent of receiver resource;
Sends an Acknowledgment for its related Cluster Agent;
}

2.2.2 Intra grid load balancing algorithm

Load balancing at this level is used if the Cluster Agent fails to balance its load among its related resources. In this case, each overloaded cluster migrates Worker Agents from its overloaded resources to underloaded clusters. In contrast to the intra-cluster level, we should consider the communication cost among clusters. Knowing the global state of each cluster, the overloaded cluster can send its Worker Agents for under-loaded clusters. The selected under-loaded clusters are those that require minimal communication cost for migrating agents from overloaded clusters. The agent can be transferred only if the sum of its latency in the source cluster and cost transfer is lower than its latency on the receiver cluster. This assumption will avoid making useless agent migration.

We associate a period to each Cluster Agent, during which each Cluster Agent sends its current load information to the other clusters. So, a Cluster Agent can receive new load information about another one at any time. This updated information will be considered in the next period.

Cluster Agent: intra-grid load balancing algorithm
Input: $Load_{cluster}$

```
Output: Sender Resource, Receiver Resource
{
If (events_happens () =7) then
{
Receives Load_cluster of other clusters of grid;
Collects Load_cluster in the table of clusters information
Sort table of clusters information by ascending order relative to their load
Select the first cluster as receiver cluster;
Sorts the resources of receiver cluster by ascending order of their load
Receiver Resource = the first resource of list of resources in receiver cluster
Sorts the resources of current cluster by descending order of their load
Sender Resource = the first resource of list of resources in sender cluster
Sorts Worker Agents of first resource of current cluster by selection policy and communication cost;
Sends this information for the Migration Agent
}
```

3. Implementation

We implemented a system prototype using JADE [6] (Java Agent Development Framework) for agent implementation, and Alea 2 [7] (Job Scheduling Simulator based on GridSim) as a simulator of the grid. Alea 2 is based on GridSim Toolkit [8] and represents an extension that contains better tools for scheduling algorithm implementation visualization competency and an upper speed of simulations.

To find the constant H for calculating the higher threshold, we execute our load balancing method 10 times for the different values of H = 0.5, 0.6, 0.7, 0.8, 0.9, and observed number of migration.

Figure 3 demonstrates the number of migration for the different values of H threshold. The number of migration augments with H threshold values but when the value of H threshold changes to 0.9 and 1, the number of migration is augmenting intensely. So we set the value of H threshold at 0.9.

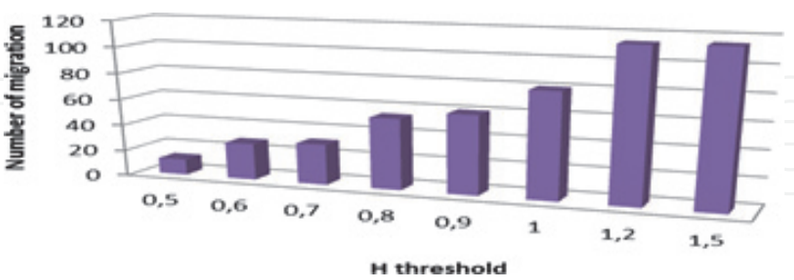

Figure 3.
Number of migration on different value of H.

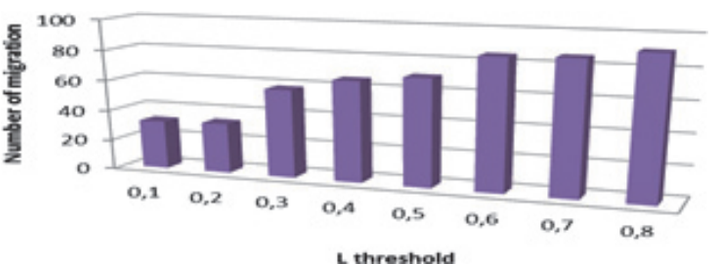

Figure 4.
Number of migration on different value of L.

In **Figure 4**, we are using the H threshold = 0.9. So, to find the best value of the lower threshold we executed our load balancing algorithm 10 times for the different values of the L threshold L = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and we observed the number of migration. We found that number of migration augmented with the value of L threshold. It offers the best result at L = 0.3.

3.1 Workload

We modeled the complex data set from the National Grid of Czech republic MetaCentrum, these data permit us to implement very realistic simulations. Also, it offers information about machine failures and specific job requirements and that information influences the quality of solutions generated by the scheduling algorithms. Job description includes (Job ID, user, queue, number processors used, etc.). Also, the description of clusters includes complete information with RAM size, CPU speed, CPU architecture, operating system, and the list of supported properties (cluster location, allowed queue (s), and network interface, etc.). Additionally, information machines were in maintenance (failure/restart). Finally, the list of queues containing their time limits and priorities is provided. More details on the trace file used can be found at [9].

3.2 The simulation environment

A class library was developed that simulates the activities of an agent platform. This library, called ABLBM (Agent-based load balancing), includes the classes: Grid Agent, Cluster Agent, Migration Agent, LBC Agent, and Resource Agent. The simulation is initialized by the Grid Agent class which makes instances of resources, jobs, and other entities as required by the GridSim standard. Grid Agent reads information describing the grid resources and jobs from a data file, reads the jobs from the data_set file, and dynamically produces the job instances over time. **Figure 5** specifies the Grid Resource parameters such as resource ID, resource's CPU speed, and resource's memory capacity. Next, Grid Agent lists all the available grid resources within the grid environment. When the simulation time is equal to the job submission time, the Grid Agent starts the Cluster Agent and dynamically sends the jobs created for the Resource Agents over time.

Based on its own load and the estimated balance threshold, Cluster Agent analyzes the load state of the cluster. In the imbalance state, Cluster Agent defines the

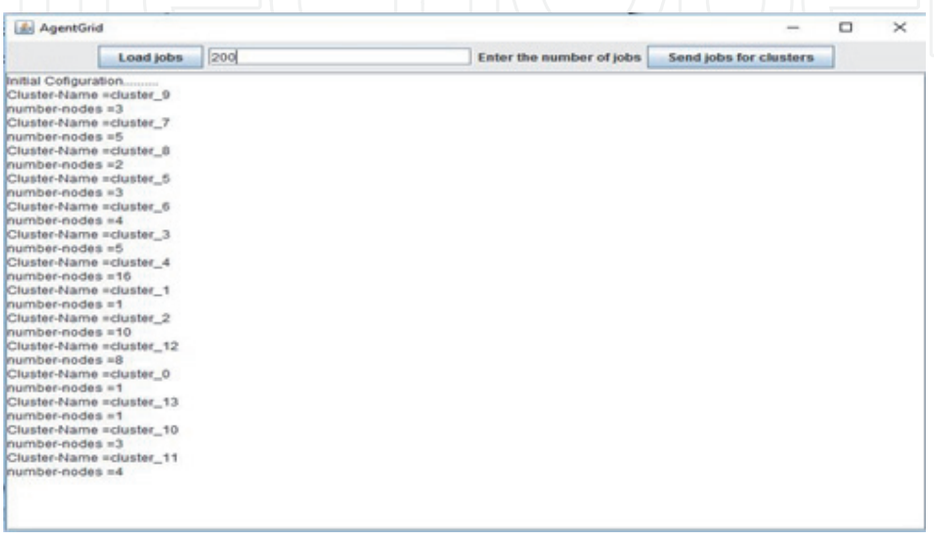

Figure 5.
Grid agent interface.

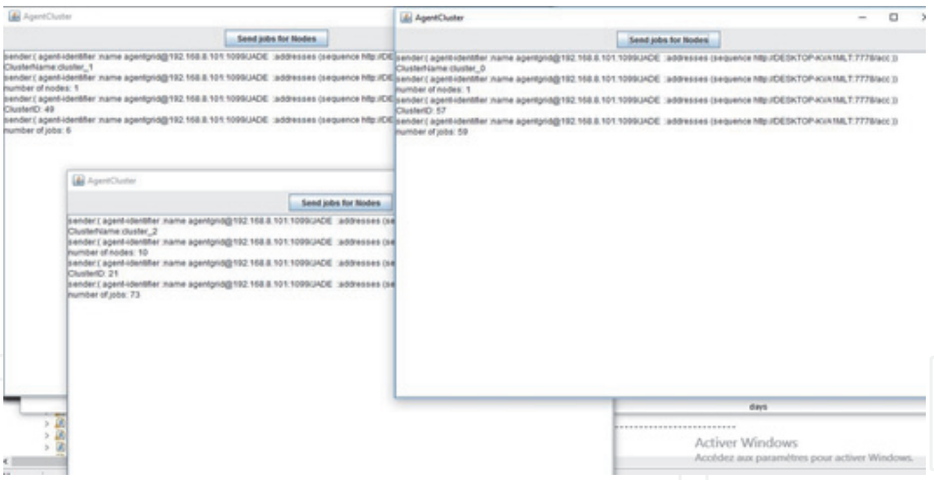

Figure 6.
Cluster agent interface.

overloaded resources (sources) and the underloaded ones (receivers), depending on their load information by using the threshold values. Grid Agent and Cluster Agent interfaces are shown in **Figures 5** and **6**.

3.3 Model comparison with some works

Model	System configuration	Load information gathering policy	Selection policy	Location policy	Decision making	Migration condition	Implementation
ABLBM	The system contains a set of computing resources hierarchy of control, with six types of agents	Event-based information gathering	Credit -based concept	receiver resource is the most under loaded and it has the highest communication load with the migrated Agent	Migration decision is taken by Cluster Agent	Cluster state is unbalanced;	Jade [6] + Alea2 [7] simulator
VM, dynamic balancing [10]	The system contains entity, federate, VM, and host with migration management agent	periodic-based information gathering	computation and communication cost	receiver resource is the least loaded host	Migration decision is taken by migration management agent	host is over-loaded,	AST-RTI [11] version 2.0 + C++
LB in distributed MAS [4]	The system contains a set of nodes decentralized in control, with seven agents	Event-based information gathering	Credit -based concept	The destination node is the node with the least LC (Location Credit) value	Migration decision is taken locally by the LBC Agent	Local load value is greater than the load threshold value	Java + Jade
A2LB [12]	The system contains a set of VMs centralized in control, with three agents	periodic-based information gathering	Not cited	receiver VM having desired configuration	Migration decision is taken by Load agent	fitness value of a VM becomes less than or equal to threshold value	Java
N LB WITH STRONG MIGR ATION IN AN	The system contains a set of machines centralized in	periodic-based information gathering	task cost	receiver is available worker agent in	Migration decision is taken by	network traffic analysis	Jade

Model	System configuration	Load information gathering policy	Selection policy	Location policy	Decision making	Migration condition	Implementation
AGENT BASED GRID SYSTEM USING CSP APPROACH [13]	control, with five agents			the desired container	migration manager		

4. Conclusion

Recognizing key factors to prove the convergence of grid and MAS and models is not a simple task. We note that the current state of GRID and MAS research activities are necessarily developed to enable justifying the study of the path towards an integration of the two fields.

We have presented a theoretical comparison between some related works and the proposed model. The proposed model has some unique features. It is hierarchical, which facilitates the circulation of information through the tree and defines the flow of messages between agents. Also, the proposed Agent-based load balancing model uses an event-driven information gathering policy, the latter being especially beneficial in terms of economy of usage of network resources. Furthermore, it can achieve excellent performance with significantly less computational load and system instability than a periodic information gathering policy. To select the migrating agent, we use the credit-based concept, accordingly, some factors are considered to calculate the credit value. Moreover, in the selection of receiver resources, we take into consideration the resource loads and the communication between the receiver resources and the migrating agent for avoiding the migration for external resources and reducing the communication cost. The migration decision is taken locally by Cluster Agent, where each cluster agent to balance its load among its associated resources. If it fails, the Cluster Agent migrates worker agents to underloaded clusters based on the load information received by other clusters. Finally, it supports flexibility and expandability, thus, various intelligent agents have been deployed to decrease system complexity by modularization. Moreover, it is easy to modify its components, and add more features and functions to it.

In theory, the multi-agent architecture of load balancing systems introduces important improvements, such as better average performance when one computer is not working and a lower system-error probability. In terms of the development process, fault-tolerance, and scalability, the agent approach offered the expected improvements, both in objective real-world measurements and in the subjective observations of designers, developers, and users.

On another hand, we could not overcome several well-known problems when designing distributed systems. For example, handling failed entities, synchronization problems, and query-response-related issues turned out to be the same as in any distributed programming. It is important to be aware of the advantages and disadvantages of the agent and non-agent approaches, but the most important point is whether the advantages prevail. For load balancing systems, our theoretical analysis and practical experiences both indicate that the advantages of agent-based load balancing systems clearly be more than the observed disadvantages.

The system performance was not studied yet. Thus, there is a need to analyze execution efficiency and compare it to available Agent-based load balancing platform evaluations. Further research is going to concentrate on execution performance.

IntechOpen

Author details

Wided Ali* and Fatima Bouakkaz
Larbi Tebessi University, Tebessa, Algeria

*Address all correspondence to: wided.ali@univ-tebessa.dz

IntechOpen

References

[1] A. Wided, K. Okba, and B. Fatima. Load balancing with Job Migration Algorithm for improving performance on grid computing: Experimental Results. Advances in Distributed Computing and Artificial Intelligence Journal, 2019; 8(4): 5–18. DOI:10.14201/ADCAIJ201984518

[2] A. Wided, K. Okba . A new agent based load balancing model for improving the grid performance. Multiagent and Grid Systems Journal, 2020; 16(2):153–170. DOI: 10.3233/MGS-200326.

[3] Chow K-P, Kwok Y-K. On load balancing for distributed multiagent computing. IEEE Trans Parallel Distrib Syst ,2002;13(8):1153–61.

[4] A.M. Metawei, G. Salma,M.H. Sahar , M.N.Salwa .'Load balancing in distributed multi-agent computing systems', Ain Shams Engineering Journal,2012, 3(3):237–249.

[5] R. Gupta . Coding categorical variables in regression models: dummy and effect coding.Cornell Statistical Consulting Unit,2008,4(2):202–210.

[6] F.L. Bellifemine, G.Caire, and D. Greenwood, Developing Multi-Agent Systems with JADE. John Wi-ley & Sons,2007, NJ. ISBN: 978–0–470-05747-6

[7] D. Klusáček and H.Rudová. Alea 2 job scheduling simulator. In Proceedings of the 3rd International Conference on Simulation Tools and Techniques (SIMUTools 2010), Torremolinos, MalaMA, Spain. DOI: 10.4108/ICST.SIMUTOOLS2010.8722

[8] R.Buyya, and M.Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, The Journal of Concurrency and Computation: Practice and Experience (CCPE),2002, 14:13–15. DOI: 10.1002/cpe.710

[9] N. Li, X.-Y. Peng, M.-H. Zhang, M. Wang, and G.-H. Gong, Multimedia communication over HLA/RTI, SimulationModelling Practice and Theory,2006,14(2): 161–176. https://doi.org/10.1016/j.simpat.2005.03.003

[10] A.Singh,D. Juneja, M.Malhotra. Autonomous agent based load balancing algorithm in cloud computing. International Conference on Advanced Computing Technologies and Applications, 2015. DOI: 10.1016/j.procs.2015.03.168

[11] X. Song, Y. Ma, D.Teng. A load balancing scheme using federate migration based on virtual machines for cloud simulations, Mathematical Problems in Engineering, 2015,pp.1–11. https://doi.org/10.1155/2015/506432

[12] Z. A .Sayar and N.Erdogan. Network load balancing with strong migration in an agent based grid using CSP Approach. International Journal of Grid Computing & Applications (IJGCA),2012, 3(4),43–53. DOI: 10.5121/ijgca.2012.3404

[13] http://www.cs.huji.ac.il/labs/parallel/workload/l_metacentrum