A Mobile Agent Team Works based on Load-Balancing Middleware for Distributed Computing Systems

Fatéma Zahra Benchara[1], Mohamed Youssfi[2]
Department of Computer Science, Laboratory SSDIA
ENSET Mohammedia, Hassan II University of Casablanca, Mohammedia, Morocco

Abstract—**The aim of this paper is to present a load balancing middleware for parallel and distributed systems. The great challenge is to balance the tasks between heterogeneous distributed nodes for parallel and distributed computing models based distributed systems, by the way to ensure HPC (High performance computing) of these models. Accordingly, the proposed middleware is based on mobile agent team work which implements an efficient method with two strategies: (i) Load balancing Strategy that determines the node tasks assignment based on node performance, and (ii) Rebalancing Strategy that detects the unbalanced nodes and enables tasks migration. The paper focuses on the proposed middleware and its cooperative mobile agent team work model strategies to dynamically balance the nodes, and scale up distributed computing systems. Indeed, some experimental results that highlight the performance and efficiency of the proposed middleware are presented.**

Keywords—Load balancing; middleware; parallel and distributed systems; parallel and distributed computing models; high performance computing; mobile agents; distributed computing

I. Introduction

Distributed systems play a great role by providing a promising distributed computing environment for big data applications, in order to meet their requirements, and ensure the HPC. Therefore, distributed systems have been introduced as a promising solution for HPC thanks to two main features: interconnection network speed such as: Ethernet, 4G, 5G and, their effective Middleware such as: CORBA, RMI, AMQP. These make distributed systems as a cooperative parallel and distributed environment able to implement parallel and distributed computing models, and ensure the collaboration of heterogeneous machines in order to achieve the processing power required by big data [1] applications, and reduce the computing time.

Consider the two principal distributed system challenges that these applications have to deal with. Their scalability and efficiency depends on their ability to manage the message passing paradigm, and the heterogeneity of node performance. For example, performing an application of big data classification based parallel and distributed computing models. It involves a wide number of heterogeneous distributed computing system nodes to achieve the required processing power. The heterogeneity of distributed system nodes influences negatively the performance of these models if unbalanced task assignment is performed between nodes. Therefore, an effective task assignment strategy is required to deal with the load balancing challenge.

In this paper, a new load balancing middleware is proposed, which is based on mobile agents, and implements effective method for task assignment and migration. Besides, the proposed middleware integrates a cooperative mobile agent team work model, which elaborates well defined load balancing strategies to balance the distributed computing system. Consider the great challenge of nodes performance heterogeneity in parallel and distributed systems. We will present a load balancing model that achieves these requirements. This paper is organized as follows:

- The middleware and its innovative components for load balancing process are presented in Section 3.
- The Section 4 is focused on presenting the method used by the mobile agent team work in order to elaborate load balancing strategies.
- The efficiency of proposed load balancing middleware is demonstrated, by performing an SPMD application in parallel and distributed computing system (Section 5).

II. Background

To set the scene of this paper, we begin with a brief overview of distributed computing systems [2], and their ability to perform HPC application based on parallel and distributed computing models. Consider this application is composed by a set of NT tasks $T_k\{k=1,\ldots,NT\}$, which is executed in parallel and distributed computing system of n nodes $N_i\{i=1,\ldots,n\}$. In case of homogenous system, the same number of tasks (load LB) is assigned for each node Ni with $LB_i = \frac{NT}{n}$. Otherwise, the load LB depends on the node's performance index NPI_i, by means that for each node Ni the assigned load LBi is given by:

$$LB_i = \frac{NT}{n} \times NPI_i \qquad (1)$$

Therefore, a load balancing method for HPC applications based distributed system is required. This method has to take into account the node performance index NPI_i, and grants the same computation time for all nodes. Thus, the computation time Ω^{MIN} of node N_i^{MIN} (slowest node) is equal to Ω^{MAX} of node N_j^{MAX} (faster node) with $i \neq j$.

The Mobile agents [3],[4] have impressive skills, such as asynchronous communication ability, autonomy, adaptability, and mobility. They can move from overloaded nodes to under loaded ones, and perform a balanced system. The agent's mobility can be an effective mechanism for dynamic load balancing of the system. Additionally, the mobile agents cooperate asynchronously by exchanging messages, which significantly reduces the load balancing strategy time. Further, their adaptability skill makes the proposed middleware flexible with different distributed computing systems. Thus, the mobile agents ensure effective features for scalable load balancing middleware.

III. Proposed Load Balancing Middleware

A. Middleware Overview

The proposed middleware (Fig. 1) implements a load balancing method, which behaves when an application is deployed, and performed in the system. Once, it is deployed this method defines the initial performance index by getting the node's performance capabilities. If the metadata of task is known, the defined index is used to estimate the load assignment LB_i for each node N_i. Else, the initial performance index will be used for running the application. For an iterative application, before the next iteration, this method decides the required load migration. Thus, the middleware can balance the node's load effectively.

B. Aspect based Load-balancing Middleware

The proposed middleware integrates the LoadBalancer aspect to the system. This aspect is based on AOP (Aspect Oriented Programming) approach [5],[6], which is useful for separating the functional aspects from the technical ones in an application, and allows to dynamically modify the program behavior. To do so, the middleware adds two aspects (behaviors) to the system, by the way that it can get metadata and provide the results needed to balance the system. These behaviors are described as follows:

- Load Assignment Aspect This aspect is performed when an application is deployed in order to get the load assignment of nodes.
- Rebalancing Aspect This aspect is executed when an application is running in order to get the required load migration, and rebalance the system before performing the next iteration.

IV. Effective Load Balancing Method

The proposed middleware implements an effective load balancing method (Fig. 2) for distributed computing system. This is done according to three method's main step; Initial nodes performance Determination, Load Assignment Prediction, Load Rebalancing. This method is implemented on cooperative mobile agent team works composed by two principal agents: Team Load Balancer Agent (TLBA agent), and Team Node Performance Monitor Agents (TNPMA agents) one per node. Main steps of this method are detailed as follows:

Step 1. Initial nodes performance Determination

- Metadata MDT_0 Determination
- Computing the initial performance index NPI^{T0}
- Computing the initial load LB^{T0}

Step 2. Load Assignment Prediction

- Metadata MDT_k Determination
- Computing the Node Performance index NPI^{Tk}
- Computing the initial load LB^{Tk}

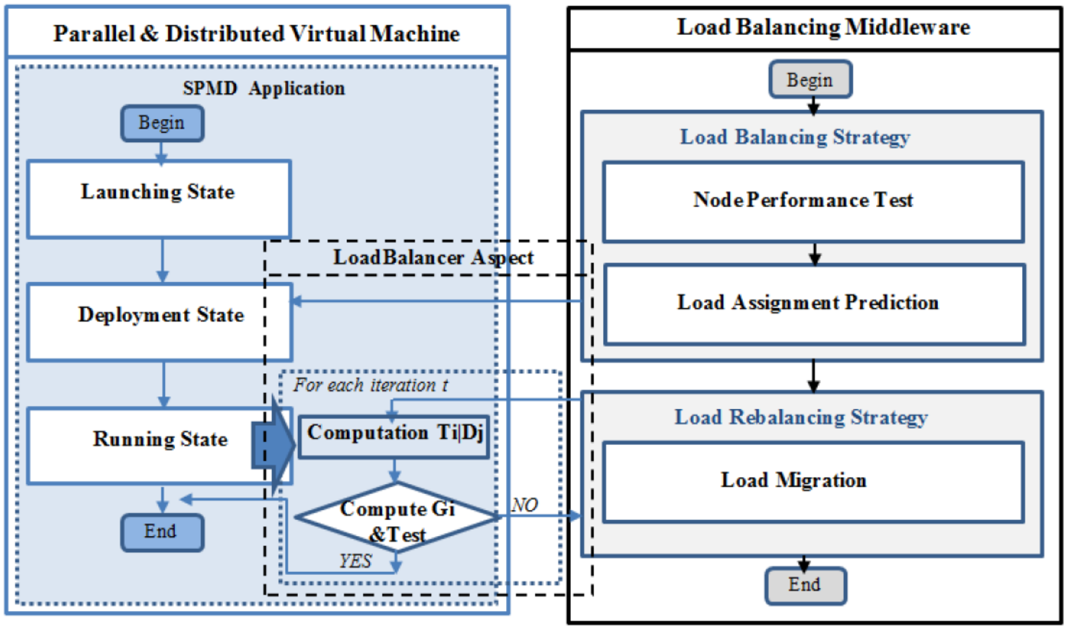

Fig. 1. Load Balancing Middleware Architecture.

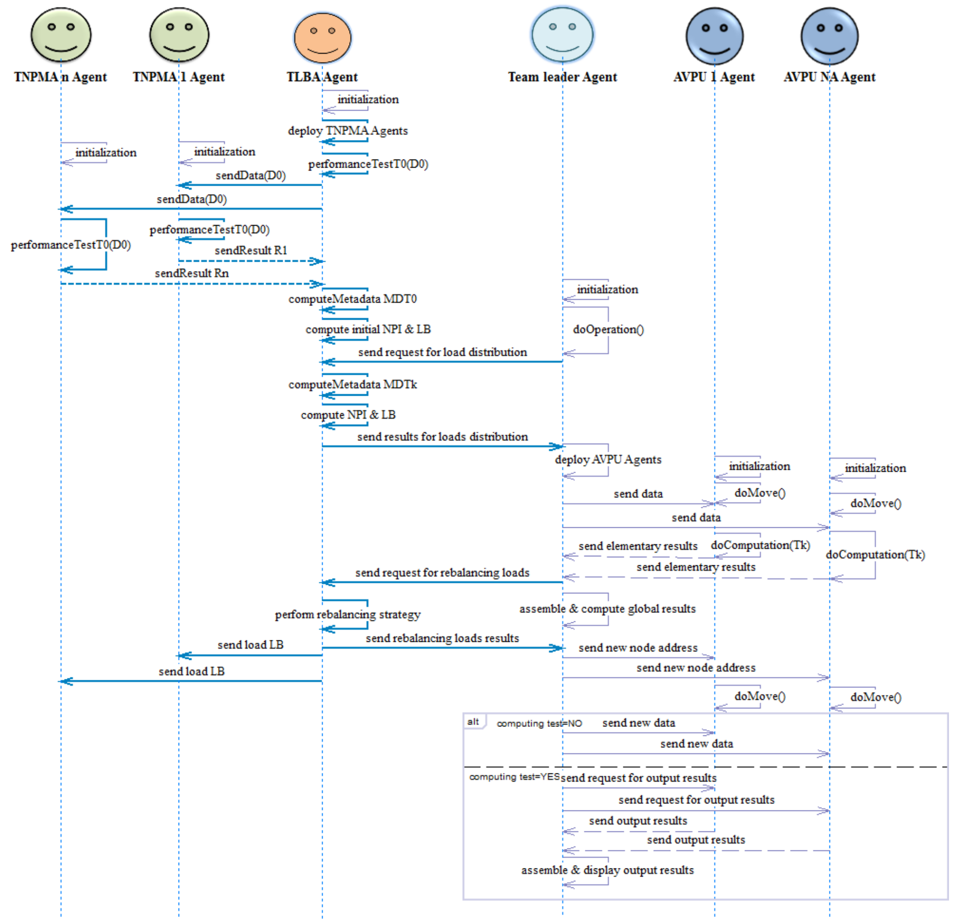

Fig. 2. Sequence Diagram of Mobile Agent Team Works Load-Balancing Method.

Step 3. Load Rebalancing

The load balancing method's steps are detailed as follows:

Step 1. Initial nodes performance Determination

- Metadata MDT_0 Determination

a) The TLBA agent deploys TNPMA(i) agent for each node N_i, after its initialization by the task T_0(initial performance task with complexity C_0) and the data D_0(initial data of size x_0).

b) The TLBA agent executes the performance test of its node N_0.

c) The TLBA agent sends the data D_0 for each TNPMA(i) agent.

d) Each TNPMA(i) agent gets the data D_0 and executes the task T_0 on data D_0.

e) Each TNPMA(i) agent returns the results R_i (the size of result y_0, and the execution time $\theta t_i^{T_0}$) to the TLBA agent.

f) The TLBA agent receives the result R_i from each TNPMA(i) agent at $t_1(i)$, and computes the communication latency $\theta l_i^{T_0}$ between the node N_i and N_0 by:

$$\theta l_i^{T_0} = \omega_i^{T_0} - \theta t_i^{T_0} \quad (2)$$

Where: $\boldsymbol{\omega_i^{T_0}}$ computation time of the task T_0 in the node N_i, which is given by:

$$\omega_i^{T_0} = t_1(i) - t_0 \quad (3)$$

- Computing the initial performance index NPI^{T0}

The TLBA agent computes the initial performance $NPI_i^{T_0}$of each node N_i by:

$$NPI_i^{T_0} = \frac{MIN(\omega_i^{T_0})}{\omega_i^{T_0}} \quad (4)$$

- Computing the initial load LB^{T0}

The TLBA agent determines the initial load LB^{T0} of each node N_i by:

$$LB_i^{T_0} = LB_{ref} \times \frac{NPI_i^{T_0}}{\langle NPI_i^{T_0} \rangle} \quad (5)$$

Where:

LB_{ref} The referenced load of each node N_i in homogeneous distributed system, which is computed by (6), where NT is the total number of tasks, and n the total number of nodes.

$$LB_{ref} = \frac{NT}{n} \quad (6)$$

$\langle \boldsymbol{NPI_i^{T_0}} \rangle$ The average of node performance index NPI_i that is computed by.

$$\langle NPI_i^{T_0} \rangle = \frac{\sum_{i=0}^{n-1} NPI_i^{T_0}}{n} \tag{7}$$

Step 2. Load Assignment Prediction

- Metadata MDT_k Determination

The TLBA agent predicts the execution time $\theta t_i^{T_k}$, and the communication latency $\theta l_i^{T_k}$, and the computation time $\omega_i^{T_k}$, respectively by:

$$\theta t_i^{T_k} = \frac{\theta t_i^{T_0} \times C_k}{C_0} \tag{8}$$

$$\theta l_i^{T_k} = \frac{\theta l_i^{T_0} \times Z_k}{Z_0} \tag{9}$$

$$\omega_i^{T_k} = \theta t_i^{T_k} + \theta l_i^{T_k} \tag{10}$$

- Computing the Node Performance index NPI^{Tk}

The TLBA agent gets the computed value of $\omega_i^{T_k}$ i, and computes the node performance index NPI_i^{Tk} by:

$$NPI_i^{T_k} = \frac{MIN(\omega_i^{T_k})}{\omega_i^{T_k}} \tag{11}$$

- Computing the initial load LB^{Tk}

The TLBA agent gets the node performance index $NPI_i^{T_k}$, and computes the load assignment LB_i by :

$$LB_i^{T_k} = \text{LB}_{\text{ref}} \times \frac{NPI_i^{T_k}}{\langle NPI_i^{T_k} \rangle} \tag{12}$$

Where :

$\langle \text{NPI}_i^{\text{Tk}} \rangle$ The average of node performance index NPI_i that is computed by:

$$\langle NPI_i^{T_k} \rangle = \frac{\sum_{i=0}^{n-1} NPI_i^{T_k}}{n} \tag{13}$$

Step 3. Load Rebalancing

1) The TLBA agent computes the experimental computation time ω_i^{EXP} by:

$$\omega_i^{EXP}(t) = \frac{\Omega_i^{EXP}(t)}{LB_i(t-1)} \tag{14}$$

2) The TLBA agent computes the new performance index NPI_i^{EXP} by:

$$NPI_i^{EXP} = \frac{MIN(\omega_i^{EXP}(t))}{\omega_i^{EXP}(t)} \tag{15}$$

3) The TLBA agent computes the new load $\text{LB}_i^{\text{EXP}}(t)$ by:

$$LB_i^{EXP}(t) = LB_{ref} \times \frac{NPI_i^{EXP}(t)}{\langle NPI_i^{EXP}(t) \rangle} \tag{16}$$

4) The TLBA agent tests the overload ΔLB_i by:

$$\Delta LB_i = LB_i^{EXP}(t) - LB_i^{EXP}(t-1) \tag{17}$$

$$\begin{cases} ON_i = N_i & if \quad \Delta LB_i < 0 \\ UN_i = N_i & if \quad \Delta LB_i > 0 \\ NN_i = N_i & if \quad \Delta LB_i = 0 \end{cases} \tag{18}$$

5) The TLBA agent determines the required load migration by running the given algorithm (Agent migration determination).

Algorithm Agent Migration Determination

```
1 : int overLNode;
2 : int underLNode;
3: for(int i=0;i<NO.size();i++){
4: overLNode=NO.get(i);
5:   for(int j=0;j<NU.size();j++){
6:  underLNode=NU.get(j);
7:    if(deltaLB[underLNode]>0){
8:      originMigration.add(overLNode);
9:      destinationMigration.add(underLNode);
10:     if(deltaLB[underLNode]>deltaLB[overLNode]){
11:       nbAgentsMigration.add(deltaLB[overLNode]);
12:             deltaLB[underLNode]=deltaLB[underLNode]-
deltaLB[overLNode];
13:             deltaLB[overLNode]=0;
14:             break;
15:     }
16:     else if(deltaLB[underLNode]<deltaLB[overLNode]){
17:        nbAgentsMigration.add(deltaLB[underLNode]);
18:              deltaLB[overLNode]=deltaLB[overLNode]-
deltaLB[underLNode];
19:        deltaLB[underLNode]=0;
20:     }
21:      else{
22:         nbAgentsMigration.add(deltaLB[overLNode]);
23:              deltaLB[overLNode]=0;
24:              deltaLB[underLNode]=0;
25:              break;
26:         }
27:      }
28:   }
29: }
```

The load deltaLB(overLNode) (line 3) which corresponds to the overloaded node is compared with deltaLB(underLNode) of the under loaded node. This is done, to decide the required load migration with the node destination. When deltaLB(underLNode) is greater than deltaLB(overLNode) (line 10), the load will move to the under loaded node (line 11). At the end, this algorithm provides the three output results:

1) originMigration list of nodes from where the load will move.

2) destinationMigration list of nodes that will receive the load.

3) nbAgentsMigration list of agents load that will move from their origin node to the appropriate destination node.

V. Results and Discussion

The proposed middleware is integrated in the parallel and distributed virtual machine [7], which is constituted by distributed computing system of 10 heterogeneous nodes. To do so, an SPMD application is chosen in order to perform the image processing of ne × me = (20×50) elementary images of size (1024×768) pixels. Thus, NA=1000 of AVPU(Agent Virtual Processing Units) agents have to execute the same task

T_k at the same time in the system. To illustrate the effectiveness feature of this middleware two case studies are considered:

Case 1 Task assignment by initial performance test

In this case the task assignment is performed by using the LB^{T0} in Table I. The initial performance test is executed by using task T_0 of complexity $C_0(x)=O(x^3)$ and data D_0 (matrix (80 × 80) where $x_0=6400$ and $y_0=6401$). For example in Table III, the value of ΔLB_8 at the node N_8 is equal to -5. This means that the node is overloaded by 5 agents, which have to move to under loaded nodes given in Table V.

Case 2 Task assignment by prediction using the metadata $MDT_{K.}$

The node task assignment is performed by using the LB^{Tk} in Table II. The predicted LB^{Tk} is based on the metadata of task T_k (complexity $C_k(x)=O(x^2)$, and data size ($x_k=786432$, $y_k=786433$), and the metadata MDT_0. In this case the load $\Delta LB_8=LB_8^{EXP(t)} - LB_8^{EXP(t-1)}$ of node N_8 is equal to -1 in Table IV. This means that the node is overloaded by only one agent, which has to move to under loaded nodes given in Table VI.

By comparing the two cases (Fig. 3), the system is balanced from the first iteration in case 2. Therefore, in case 1 it becomes balanced after the second iteration. This means that case 2 grants effective load balancing strategy of the system at the first iteration. The Fig. 4 presents that the system becomes balanced after performing the load rebalancing.

TABLE. I. RESULTS OF LOAD ASSIGNMENT BY INITIAL PERFORMANCE TEST

	Ni	Metadata MDT0	$\mathbf{NPI_i^{T_0}}$	$\mathbf{LB_i^{T_0}}$	$\Omega_i^{TH}(\omega_i^{T_0} * \mathbf{LB_i^{T_0}})$ (ms)		
		$\omega_i^{T_0}(ms)$	$\theta t_i^{T_0}(ms)$	$\theta l_i^{T_0}(ms)$			
	0	7826,00	7818	8	0,811014567	94,66	740843,0833
	1	7658,00	7640	18	0,828806477	96,74	740843,0833
	2	8072,00	8050	22	0,786298315	91,78	740843,0833
	3	7673,00	7640	33	0,827186237	96,55	740843,0833
	4	7249,00	7230	19	0,875569044	102,20	740843,0833
	5	6998,00	6980	18	0,906973421	105,86	740843,0833
	6	7155,00	7133	22	0,887071978	103,54	740843,0833
	7	8011,00	7980	31	0,792285607	92,48	740843,0833
	8	6347,00	6322	25	1	116,72	740843,0833
	9	7449,00	7430	19	0,852060679	99,46	740843,0833
$MIN(\omega_i^{T_0})$		6347,00	-	-	-	-	-
$MAX(\omega_i^{T_0})$		8072,00	-	-	-	-	-
SUM(LBT0)		-	-	-	-	1000,00	-
AVG(NPIT0)		-	-	-	0,85672663	-	-

TABLE. II. RESULTS OF LOAD ASSIGNMENT BY PREDICTION

	Ni	Metadata MDTk			$\mathbf{NPI_i^{T_k}}$	$\mathbf{LB_i^{T_k}}$	$\mathbf{\Omega_i^{TH}}$
		$\omega_i^{T_k}(ms)$	$\theta l_i^{T_k}(ms)$	$\theta t_i^{T_k}(ms)$			
	0	19428,016	983,040	18444,976	0,925852089	103,22	2005359,82
	1	20236,861	2211,840	18025,021	0,888846789	99,09	2005270,60
	2	21695,693	2703,360	18992,333	0,829080199	92,43	2005332,89
	3	22080,061	4055,040	18025,021	0,814647611	90,82	2005311,18
	4	19392,430	2334,720	17057,710	0,927551072	103,41	2005371,19
	5	18679,726	2211,840	16467,886	0,962940743	107,35	2005268,59
	6	19532,218	2703,360	16828,858	0,92091277	102,67	2005372,86
	7	22636,462	3809,280	18827,182	0,794623703	88,59	2005364,18
	8	17987,469	3072,000	14915,469	1	111,48	2005243,08
	9	19864,289	2334,720	17529,569	0,90551789	100,95	2005300,00
$MIN(\omega_i^{T_k})$		17987,469	-	-	-	-	-
$MAX(\omega_i^{T_k})$		22636,462	-	-	-	-	-
SUM(LBTk)		-	-	-	-	1000,00	-
AVG(NPITk)		-	-	-	0,89699729	-	-

TABLE. III. RESULTS OF LOAD REBALANCING (CASE 1)

N_i	$LB_i^{EXP}(t-1)$	$\Omega_i^{EXP}(t)$ (ms)	$\omega_i^{EXP}(t)$ (ms)	$NPI_i^{EXP}(t)$	$LB_i^{EXP}(t)$	$\Delta LB_i(t)$	State
0	95	1845662	19428,01613	0,92585209	103	8,00	Under
1	97	1962976	20236,86144	0,88884679	99	2,00	Under
2	92	1996004	21695,6928	0,8290802	92	0,00	Normal
3	97	2141766	22080,06144	0,81464761	91	-6,00	Over
4	101	1958635	19392,43008	0,92755107	103	2,00	Under
5	106	1980051	18679,72608	0,96294074	107	1,00	Under
6	104	2031351	19532,21837	0,92091277	103	-1,00	Over
7	94	2127827	22636,46208	0,7946237	90	-4,00	Over
8	116	2086546	17987,46931	1	111	-5,00	Over
9	98	1946700	19864,28928	0,90551789	101	3,00	Under

TABLE. IV. RESULTS OF LOAD REBALANCING (CASE 2)

N_i	$LB_i^{EXP}(t-1)$	$\Omega_i^{EXP}(t)$ (ms)	$\omega_i^{EXP}(t)$ (ms)	$NPI_i^{EXP}(t)$	$LB_i^{EXP}(t)$	$\Delta LB_i(t)$	State
0	103	2020514	19428,01613	0,92585209	104	1,00	Under
1	99	2043923	20236,86144	0,88884679	101	2,00	Under
2	92	1996004	21695,6928	0,8290802	92	0,00	Normal
3	91	2009286	22080,06144	0,81464761	91	0,00	Normal
4	103	1997420	19392,43008	0,92755107	103	0,00	Normal
5	107	1961371	18679,72608	0,96294074	105	-2,00	Over
6	103	2011818	19532,21837	0,92091277	103	0,00	Normal
7	90	2037282	22636,46208	0,7946237	90	0,00	Normal
8	111	1978622	17987,46931	1	110	-1,00	Over
9	101	2006293	19864,28928	0,90551789	101	0,00	Normal

TABLE. V. RESULTS OF AGENT'S MIGRATION (CASE 1)

	Origin Migration N_i	Destination Migration N_i	Nb Agents Migration	Load Migration Percentage
	3	0	6	37,50%
SUM	-	-	6	37,50%
	6	0	1	6,25%
SUM	-	-	1	6,25%
	7	0	1	6,25%
	7	1	2	12,50%
	7	4	1	6,25%
SUM	-	-	4	25,00%
	8	4	1	6,25%
	8	5	1	6,25%
	8	9	3	18,75%
SUM	-	-	5	31,25%

TABLE. VI. RESULTS OF AGENT'S MIGRATION (CASE 2)

	Origin Migration N_i	Destination Migration N_i	Nb Agents Migration	Load Migration Percentage
	5	0	1	33,33%
	5	1	1	33,33%
SUM	-	-	2	66,67%
	8	1	1	33,33%
SUM	-	-	1	33,33%

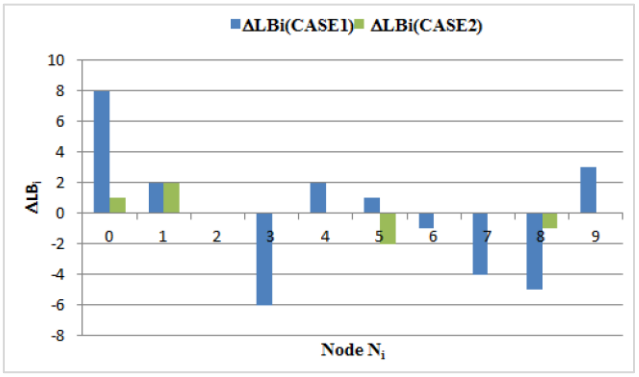

Fig. 3. Comparison of Overload in Case 1 and Case 2 of Each node N_i.

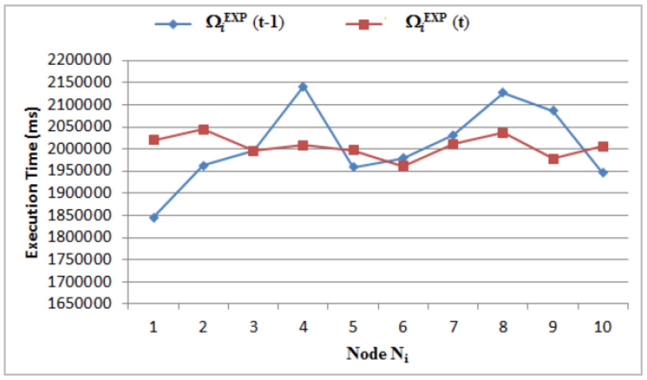

Fig. 4. Execution Time of the Task T_k before and after Load Rebalancing.

By analyzing the execution time (Fig. 5) of both cases compared with the case 3 (unbalanced system where each node receives LB_i^{NB} =100 agents), the following conclusions are achieved:

- In case 1 (Table VII) the obtained unbalance DES is DES1=$\Omega_{MAX}^{EXP}(LB_i^{T_0}) - \Omega_{MIN}^{EXP}(LB_i^{T_0})$ = 296,104s, and for case 2 is DES2=$\Omega_{MAX}^{EXP}(LB_i^{T_k}) - \Omega_{MIN}^{EXP}(LB_i^{T_k})$ = 41,2778496 s. These present the efficiency of task assignment based on prediction compared to the one based on initial performance test.
- The gain of performance φ compared to case 3, provides to the following results:
  - In case 2 the gain of performance is $\varphi 2 = \frac{\Omega_{MAX}^{EXP}(LB_i^{NB}) - \Omega_{MIN}^{EXP}(LB_i^{NB})}{\Omega_{MAX}^{EXP}(LB_i^{T_k}) - \Omega_{MIN}^{EXP}(LB_i^{T_k})}$ = 5,631 at the first iteration.
  - In case 1 the gain of performance is $\varphi 1 = \frac{\Omega_{MAX}^{EXP}(LB_i^{NB}) - \Omega_{MIN}^{EXP}(LB_i^{NB})}{\Omega_{MAX}^{EXP}(LB_i^{T_0}) - \Omega_{MIN}^{EXP}(LB_i^{T_0})}$ = 1,570 at the first iteration, which is enhanced to $\varphi' 1$= 5,631 after de second iteration.
  - The obtained gain of performance of case 2 $\varphi 2$ is equal to $\varphi' 1$, which illustrates the effectiveness of the load rebalancing step based on load migration.

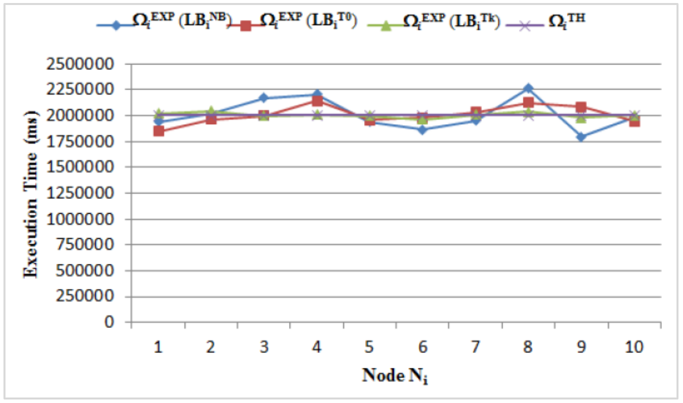

Fig. 5. Execution Time Comparison between Theoretical Execution Time Ω_i^{TH}, and Experimental One of Case 1 $\Omega_i^{EXP}(LB_i^{T_0})$ and Case 2 $\Omega_i^{EXP}(LB_i^{T_k})$ and Case 3 $\Omega_i^{EXP}(LB_i^{T_{NB}})$.

TABLE. VII. COMPARISON BASED EXECUTION TIME IN THREE CASES OF LOAD ASSIGNMENT

N_i	Ω_i^{TH}(ms)	CASE 1		CASE 2		CASE 3
		$\Omega_i^{EXP}(LB_i^{T_0})$ (ms)	$\varepsilon_i(LB_i^{T_0})$	$\Omega_i^{EXP}(LB_i^{T_k})$ (ms)	$\varepsilon_i(LB_i^{T_k})$	$\Omega_i^{EXP}(LB_i^{NB})$ (ms)
0	2005359,82	1845661,53	0,079635728	2020513,677	0,007556677	1942801,61
1	2005270,60	1962975,56	0,021091936	2043923,005	0,019275406	2023686,14
2	2005332,89	1996003,74	0,00465217	1996003,738	0,004652171	2169569,28
3	2005311,18	2141765,96	0,068046686	2009285,591	0,001981942	2208006,14
4	2005371,19	1958635,44	0,023305286	1997420,298	0,003964798	1939243,01
5	2005268,59	1980050,96	0,012575687	1961371,238	0,021891009	1867972,61
6	2005372,86	2031350,71	0,012954125	2011818,492	0,003214181	1953221,84
7	2005364,18	2127827,44	0,061067841	2037281,587	0,015916015	2263646,21
8	2005243,08	2086546,44	0,040545389	1978621,624	0,013275925	1798746,93
9	2005300,00	1946700,35	0,029222386	2006293,217	0,000495296	1986428,93

VI. Related Work

There are several inspiring load balancing approaches which have presented interesting results. Some of them are proposed for distributed systems [8], grid [9],[10], P2P systems [11],[12], and also for cloud computing systems [13],[14], and heterogeneous computing systems [15],[16],[17]. The main idea behind the considered challenge is the effective method for task assignment. To do so, the approaches in [18],[19] are based on the states of the nodes, which are grouped on domains. In [20], it is based on an index of load that is defined by the summation of the active services duration in the node; without taking into account the communication latency. The load prediction method is also investigated in load balancing models; such as [21] for dynamic load balancing of HLA based distributed simulations. However, system can further be unbalanced when running an application. This leads authors to propose dynamic load balancing algorithm based-load migration [22] in order to move the load from the overloaded nodes to the under loaded ones. It is performed by moving just one unit per iteration [23],[24],[25],[26], or a fixed number [27],[28]. Further, in [29] it's achieved by exchanging the load between neighbor nodes.

Multi-agent based load-balancing approach has been investigated as a promising paradigm for this challenge. The agent is used in [30] to represents the node for load balancing process, and in [31] it is implemented to monitor and detect the congestion in the nodes, and [32] to perform the load assignment. Further, in [33] the agent is deployed to encapsulate the tasks that will be executed.

Thereby, the mobile agents allow the load migration between nodes even in [34] homogeneous distributed system, or heterogeneous one based on node prediction algorithm. Accordingly, the proposed middleware combines two methods; task assignment according to node performance, and task migration, by the way to assign a set of tasks T_k $\{k=1,\ldots, NT\}$ to a set of $N_i\{i=1,\ldots, n\}$ nodes. Through, thanks to these several interesting works, the proposed work develops their foundation in the following ways. In this paper, a new load balancing middleware for distributed computing systems is proposed and implemented with three main focuses:

- Effective task assignment method using node performance prediction based on communication latency of each node, with integrated task migration algorithm.
- Optimized load balancing time by using the asynchronous communication mechanism between agents.
- Scalable Load balancing middleware for SPMD applications based parallel and distributed computing systems.

VII. Conclusion

The proposed load balancing middleware based cooperative mobile agents team work is a new paradigm, which is implemented using the aspect oriented approach for separating the load balancing aspect from distributed system. Through, this middleware can be integrated with different distributed computing systems for node task assignment problem. The proposed middleware deploys the mobile agent for each node in order to get the node performance. When the program is deployed the mobile agents perform an initial performance test based on referenced task in order to compute the appropriate task assignment of each node. In the case when the application tasks metadata; the complexity and the data size are known, the proposed method can predict with precision the task assignment for each node. When the application is running, if the system becomes unbalanced, the middleware executes the rebalancing algorithm to identify and decide the required load migration and rebalance the system. The obtained results, related to the execution time of each node and the gain of performance, demonstrates that the proposed middleware can ensure effective balanced distributed computing system and enhance its performance. Further, it is interesting to have a load balancing solution which handles the middleware failures. To do so, an extended work is driven in order to propose and implement a fault tolerance module for the proposed load balancing middleware.

References

[1] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. S. Netto, and R. Buyya, "Big Data computing and clouds: Trends and future directions," J. Parallel Distrib. Comput., vol. 79, pp. 3–15, 2015.

[2] A. Puder, K. Romer, and F. Pilhofer, "Distributed Systems Architecture," Morgan Kaufmann Publ., vol. 53, no. 9, pp. 1689–1699, 2006.

[3] W. R. Braun, P., & Rossak, Mobile agents: Basic concepts, mobility models, and the tracy toolkit. 2005.

[4] F. Bellifemine, G. Caire, and D. Greenwood, Developing Multi-Agent Systems with JADE. 2007.

[5] E. Hilsdale and J. Hugunin, "Advice Weaving in AspectJ," in Proceedings of the 3rd International Conference on Aspect-oriented Software Development, 2004, pp. 26–35.

[6] M. Lippert and C. V. Lopes, "A Study on Exception Detection and Handling Using Aspect-oriented Programming," in Proceedings of the 22Nd International Conference on Software Engineering, 2000, pp. 418–427.

[7] F. Z. Benchara, M. Youssfi, O. Bouattane, H. Ouajji, and M. O. Bensalah, "A New Distributed Computing Environment Based on Mobile Agents for SPMD Applications BT-Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015," 2016, pp. 353–362.

[8] Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 2, pp. 585–599, 2016.

[9] D. S. Acker and S. Kulkarni, "A dynamic load dispersion algorithm for load-balancing in a heterogeneous grid system," in 2007 IEEE Sarnoff Symposium, 2007, pp. 1–5.

[10] M. Li, P. He, and L. Zhao, "Dynamic Load Balancing Applying Water-Filling Approach in Smart Grid Systems," IEEE Internet of Things Journal, vol. 4, no. 1. pp. 247–257, 2017.

[11] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-peer Systems," in Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2004, pp. 36–43.

[12] R. Bhardwaj, V. S. Dixit, and A. K. Upadhyay, "A propound method for agent based dynamic load balancing algorithm for heterogeneous P2P systems," 2009 International Conference on Intelligent Agent & Multi-Agent Systems. pp. 1–4, 2009.

[13] H. C. Hsiao, H. Y. Chung, H. Shen, and Y. C. Chao, "Load Rebalancing for Distributed File Systems in Clouds," IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 5, pp. 951–962, 2013.

[14] J. Zhao, K. Yang, X. Wei, Y. Ding, L. Hu, and G. Xu, "A Heuristic Clustering-Based Task Deployment Approach for Load Balancing Using

Bayes Theorem in Cloud Environment," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2. pp. 305–316, 2016.

[15] J. Shen, A. L. Varbanescu, Y. Lu, P. Zou, and H. Sips, "Workload Partitioning for Accelerating Applications on Heterogeneous Platforms," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 9. pp. 2766–2780, 2016.

[16] E. Hwang, S. Kim, T. k. Yoo, J. S. Kim, S. Hwang, and Y. r. Choi, "Resource Allocation Policies for Loosely Coupled Applications in Heterogeneous Computing Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 8. pp. 2349–2362, 2016.

[17] J. Y. Jang, H. Wang, E. Kwon, J. W. Lee, and N. S. Kim, "Workload-Aware Optimal Power Allocation on Single-Chip Heterogeneous Processors," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 6. pp. 1838–1851, 2016.

[18] F. auf der Meyer Heide, B. Oesterdiekhoff, and R. Wanka, "Strongly adaptive token distribution," Algorithmica, vol. 15, no. 5, pp. 413–427, 1996.

[19] P. Berenbrink, T. Friedetzky, and R. Martin, "Dynamic Diffusion Load Balancing BT - Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings," L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1386–1398.

[20] Y. Qiao and G. v. Bochmann, "A Diffusive Load Balancing Scheme for Clustered Peer-to-Peer Systems," 2009 15th International Conference on Parallel and Distributed Systems. pp. 842–847, 2009.

[21] R. E. De Grande, A. Boukerche, and R. Alkharboush, "Time Series-Oriented Load Prediction Model and Migration Policies for Distributed Simulation Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 1. pp. 215–229, 2017.

[22] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," J. Parallel Distrib. Comput., vol. 7, no. 2, pp. 279–301, 1989.

[23] A. Cortés, A. Ripoll, F. Cedó, M. A. Senar, and E. Luque, "An asynchronous and iterative load balancing algorithm for discrete load model," J. Parallel Distrib. Comput., vol. 62, no. 12, pp. 1729–1746, 2002.

[24] Y. F. Hu and R. J. Blake, "An improved diffusion algorithm for dynamic load balancing," Parallel Comput., vol. 25, no. 4, pp. 417–444, 1999.

[25] E. Luque, A. Ripoll, A. Cortes, and T. Margalef, "A distributed diffusion method for dynamic load balancing on parallel computers," Proceedings Euromicro Workshop on Parallel and Distributed Processing. pp. 43–50, 1995.

[26] T. A. Murphy and J. G. Vaughan, "On the relative performance of diffusion and dimension exchange load balancing in hypercubes," in PDP, 1997.

[27] P. Berenbrink, T. Friedetzky, and Z. Hu, "A new analytical method for parallel, diffusion-type load balancing," J. Parallel Distrib. Comput., vol. 69, no. 1, pp. 54–61, 2009.

[28] F. Cedo, A. Cortes, A. Ripoll, M. A. Senar, and E. Luque, "The Convergence of Realistic Distributed Load-Balancing Algorithms," Theory Comput. Syst., vol. 41, no. 4, pp. 609–618, 2007.

[29] I. Konstantinou, D. Tsoumakos, and N. Koziris, "Fast and Cost-Effective Online Load-Balancing in Distributed Range-Queriable Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 8. pp. 1350–1364, 2011.

[30] S. Banerjee and J. P. Hecker, "A Multi-agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing BT - First Complex Systems Digital Campus World E-Conference 2015," P. Bourgine, P. Collet, and P. Parrend, Eds. Cham: Springer International Publishing, 2017, pp. 41–54.

[31] X.-J. Shen et al., "Achieving dynamic load balancing through mobile agents in small world P2P networks," Comput. Networks, vol. 75, pp. 134–148, 2014.

[32] O. Rihawi, Y. Secq, and P. Mathieu, "Load-Balancing for Large Scale Situated Agent-based Simulations," Procedia Comput. Sci., vol. 51, pp. 90–99, 2015.

[33] S. Hunt, Q. Meng, C. Hinde, and T. Huang, "A Consensus-Based Grouping Algorithm for Multi-agent Cooperative Task Allocation with Complex Requirements," Cognit. Comput., vol. 6, no. 3, pp. 338–350, Sep. 2014.

[34] J. Liu, X. Jin, and Y. Wang, "Agent-based load balancing on homogeneous minigrids: macroscopic modeling and characterization," IEEE Trans. Parallel Distrib. Syst., vol. 16, no. 7, pp. 586–598, 2005.