

# **Load Balancing Algorithm Based on Aspect Oriented Approach for a Multi Agent Distributed System**

**Mohamed Youssfi, Omar Bouattane and Abdelaziz Daaif**

Lab. SSDIA, ENSET, University Hassan II of Casablanca, Morocco

**Mohammed Ouadi Bensalah**

FSR, University Mohammed V AGDAL Rabat, Morocco

Copyright © 2015 Mohamed Youssfi et al. This article is distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

## **Abstract**

In this paper, we present a new load balancing system based on aspect oriented approach for multi agents distributed systems. The proposed model is based on using a dispatcher agent assigned to dynamically distribute tasks received in random requests coming from agent producers. We suppose that tasks are assigned to be executed by workers agents deployed over the nodes of the distributed system. To make the initial mechanism of task distribution balanced, we propose a separated aspect where we propose a load balancing procedure assigned to be affected to the dispatcher by an aspect weaving in compilation or run times. The load balancing aspect increases the performances of the dispatcher agent using code advices in different join points. The task distribution code advice is used to implement the load balancing algorithm which determines the under loaded worker agent who will be assigned to execute the current task  $T_i$ . In this algorithm, we take into account the heterogeneity of the processing performances, the communication latency, occupancy rate of each node and the complexity of the current task. In this model we can also use metadata of the tasks to estimate more accurately the required times to perform each task. This mechanism allows predicting the performance offered by the distributed system according to the selected computational model. To describe the proposed approach, some communication diagrams, the established mathematical model and an application example are presented to prove the effectiveness of the proposed model.

**Keywords:** Parallel and Distributed Computing, Multi Agent System, Load balancing algorithm, Aspect Oriented Approach, High Performance Computing

## 1 Introduction

In the high performance computing domain, we need to combine the performance of several processing and storage units to improve the execution performances of the applications. Parallel and distributed models became the natural solutions for several types of computational systems [1] [2] [3]. In such systems, the authors assume that the tasks are randomly assigned to the different nodes of the distributed system, and create an unbalanced load problem at the nodes. The performance of the middleware used by the distributed in a distributed system depends mainly on the quality the load balancing algorithm used to distribute tasks to the heterogeneous nodes of the distributed system. Indeed, if the association of multiple machines to distribute intensive computations using large amounts of data is required, it is important to notify that the heterogeneity of the associated machines must be taken into account during the task assignment phase. It is therefore necessary to know, statically and dynamically, the performance of each node of the grid in terms of computing speed, storage capacity, throughput of the network connection and occupancy rate of the machine. It is well known that even if the physical features of the computer machines are fixed and remain unchanged, some other features of the grid may change dynamically and can be known just at runtime. These features may be: the current occupancy rate of the processing units, the current speed of the network connection and working conditions of each node. In the literature many middleware have been designed for distributed computing, but their performances depend on the quality of the used load balancing system. In [4], Authors propose a framework of parallel remote sensing image interpretation in desktop grid. This framework consists of modules such as job partitioning and scheduling, load balancing, communication, and image processing. In this work they also provided an implementation based on java multi threaded programming. In the same way, several load balancing algorithms, designed for distributed systems, have been proposed and reviewed [5] [6] [7]. In the distributed systems, the tasks assignment is a dynamic process that can sometimes be unpredictable. This indicates that the load state of nodes cannot be static. Subsequently, a dynamic load distribution strategy is necessary. The popular load balancing approach introduced in [8], is based on the diffusion concept. This algorithm acts iteratively to balance loads in the nodes of the distributed system. The main idea behind this algorithm is that for each iteration, overloaded nodes exchange individually their loads with their under loaded neighbors. The diffusion algorithm [9, 10] collects information data from a set of nodes grouped into domains. The algorithm tries to balance the loads inside each domain. To do so, the excess loads are transferred to the under loaded nodes in their own domain, thus reducing communication overhead. In [11], the two following techniques are used to achieve load-balancing state, they are: iterative key redistribution between neighbors and node migration. The authors have proposed

a hybrid method to achieve both fast and cost-effective load-balancing in distributed systems that support range queries. In [12], Authors propose an effective load balancing algorithm for balancing loads over an entire distributed/parallel system. Their algorithm has two attractive features: it is of simplest structure and can be implemented in a decentralized fashion. Many research works have been carried out under the assumption that, each node is allowed to move just one unit of load per iteration [13, 14, 15, 16]. In other works, authors proposed to transmit a constant number of loads [17, 18] to balance the system. In a simple diffusion algorithm, each node measures the difference of loads between itself and each of its neighbors, then it sends a fixed part of its excess loads to the under loaded neighboring nodes. This strategy and others [19] were originally designed under the assumption that the load is represented by a non-negative real number, but in most cases, the load is considered as an integer number. This is the case in parallel and distributed computing systems [21, 22, 23, 24]. In [25], authors have developed a distributed load balancing algorithm for a grid computing. They used a load index defined as the sum of the service time of works currently underway in a node without taking into account communication latency effects. To carry the loads in a distributed system, the authors have used in [26] a mobile agent, which migrate the loads from overloaded nodes to under loaded ones and considering that nodes are homogeneous. In [27] the authors proposed a new algorithm for dynamic decentralized distribution which is dynamically adapted to parameters variations. The load balancing algorithm is assigned to be performed on each node. It supports load balancing for heterogeneous distributed systems.

Separation of concerns is an important software engineering principle. It refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concern. Concerns can range from high-level notations like security, transactions, persistence and quality of services to low-level notations like buffering, caching and logging. They can also be functional, such as business logics or non-functional like synchronization. Aspect-oriented programming (AOP) is a technique for improving separation of concerns in software design and implementation. AOP works by providing explicit mechanisms for capturing the structure of crosscutting concerns. AOP has been created by Xerox PARC (Palo Alto Research Center) to overcome the limitations of object-oriented programming (OOP). In [30] authors describe the implementation of advice weaving in AspectJ which picks out dynamic join points in a program's execution with point cuts and uses advice to change the behavior at join points. In [31], authors reports on a study to investigate AOP's ability to ease tangling related to exception detection and handling. In this study, authors took an existing framework written in Java TM, the JWAM framework, and partially reengineered its exception detection and handling aspects using AspectJTM, an aspect oriented programming extension to Java.

In distributed computing, the load balancing system can be considered as a separate concern of the task distribution process. In this paper, we present a new load balancing system based on aspect oriented approach for multi agents distributed system. The proposed model is based on using a dispatcher agent assigned

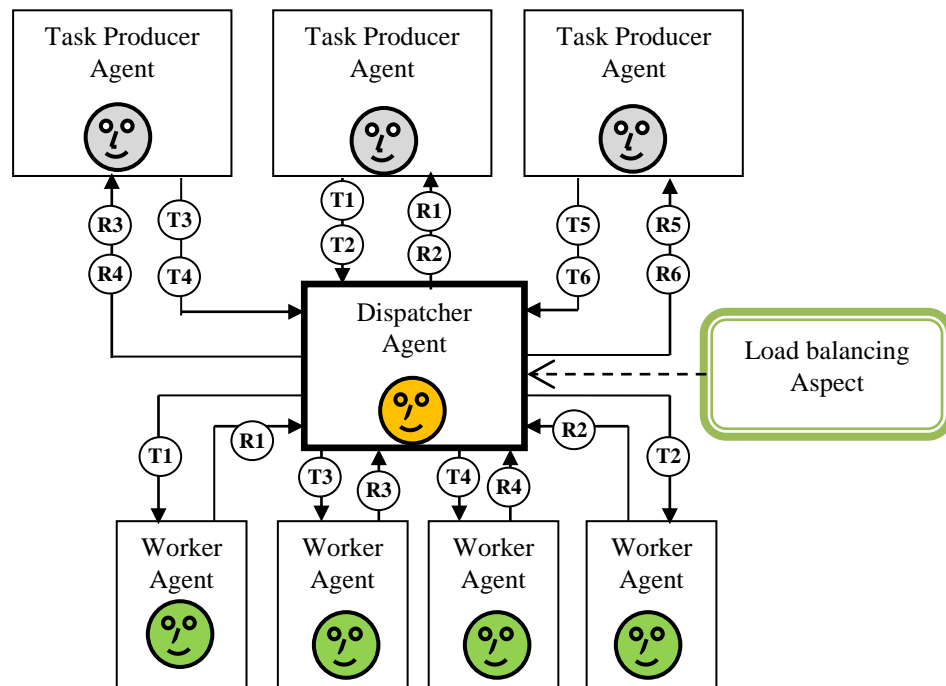
to dynamically distribute tasks enveloped in requests coming from agent producers. We suppose that tasks are assigned to be executed by workers agents deployed over the nodes of the distributed system. In this model, we also suppose that the dispatcher is initially programmed to use a non balanced algorithm in the task distribution operation. To make the initial mechanism of distribution balanced, we propose a separated aspect where we propose a load balancing procedure assigned to be affected to the dispatcher by an aspect weaving in compilation or run times. The aspect load balancing procedure is based on collecting dynamically information about the processing performance and communication latency of the different nodes of the distributed system. To initialize the load balancing system, the aspect uses a reference task executed by all agent workers in order to establish a baseline data for distributed load estimation of any given task. In this model we can also use metadata of the tasks to estimate more accurately the required times to perform each task. This mechanism allows predicting the performance offered by the distributed system according to the selected configuration of the selected computational model.

After presenting the task distribution architecture in the second section, the third section is devoted to describe the load balancing technique of the proposed approach. In the forth section, we will present an example of application where we prove the effectiveness of the proposed model. The last section gives some concluding remarks and perspectives for the future related works.

## 2. Task Distribution Architecture

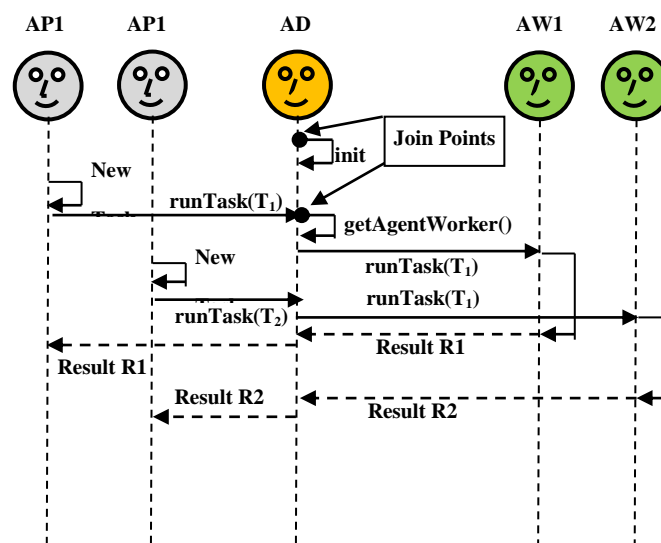
The task distribution model, presented in figure 1, is composed by 3 types of agents:

- Task Producer Agent: This agent produces tasks  $T_i$  to be executed by the distributed system. Each task is sent to the dispatcher agent in a request. The task brings the input data to be processed and the program code representing the algorithm implementation to run by an agent worker. The request is represented by an ACL message.
- The dispatcher agent: This agent receives randomly, in a queue, requests from producer agents. Each request brings a task assigned to be executed by the least loaded agent worker. Initially the dispatcher is programmed to distribute tasks without using any load balancing algorithm. In this situation, the dispatcher distribute tasks using a simplest procedure without taking in account the heterogeneity of the tasks, nodes of the distributed system and the communication latency. In this situation, the system is not load balanced. To bring the system into a load balance state, we propose a separate aspect that grafted the dispatcher agent a load balancing behaviour.
- The worker agent: This agent is appointed to perform the tasks  $T_i$  assigned to it by the dispatcher agent. When the task is performed, the result  $R_i$  is sent to dispatcher in an ACL message. This later is delivered to the agent producer.



**Figure 1.** The task distribution model

The figure 2 represents a sequence diagram where we show 2 producer agents *AP1* and *AP2* which produce respectively new tasks  $T_1$  and  $T_2$ . The two tasks are sent to the dispatcher agent. This later execute its local procedure *getAgentWorker* to determine an agent worker which will be assigned to execute the current task. This operation will be used by the load balancing aspect as a code advice. In this example, the tasks  $T_1$  and  $T_2$  are respectively assigned to the worker agents *AW1* and *AW2*. The corresponding results *R1* and *R2* are returned to the dispatcher. This later return results to the corresponding agent producers.



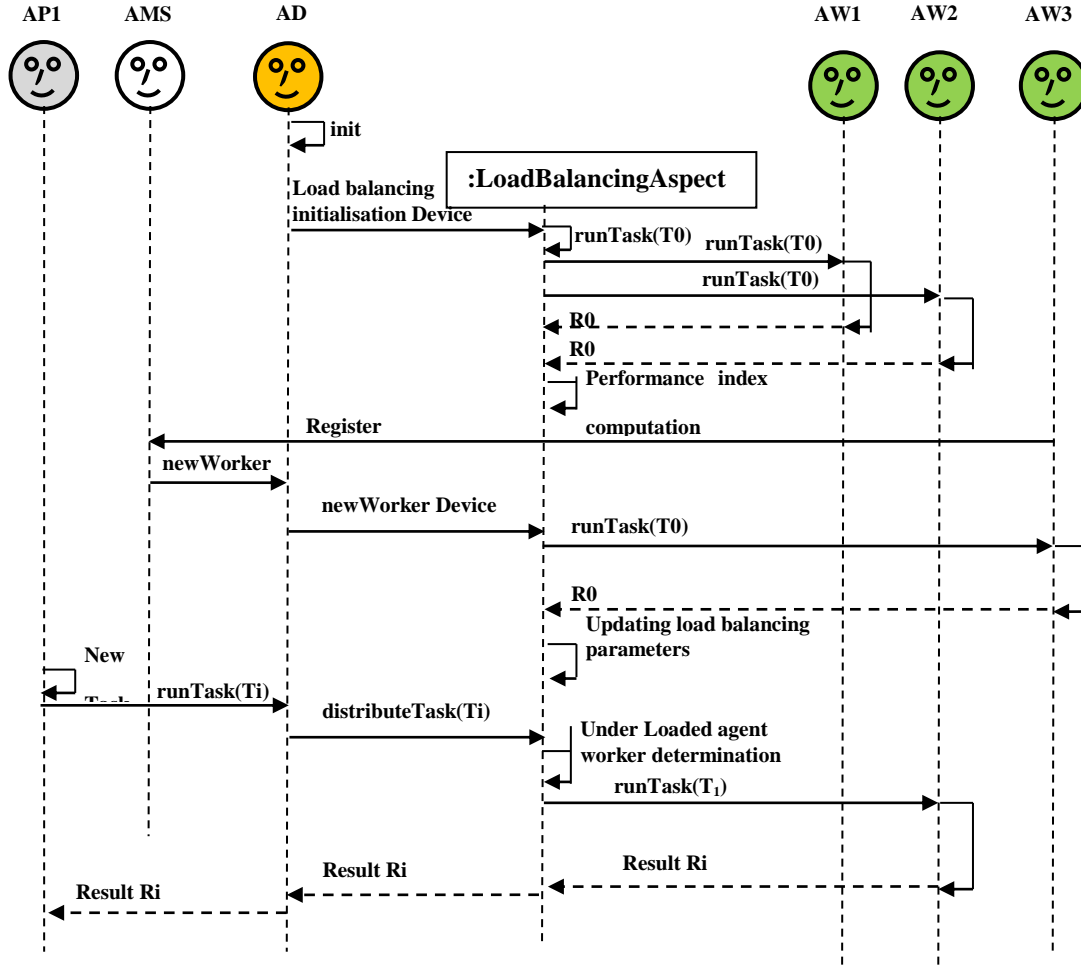
**Figure 2.** The task distribution model without load balancing aspect

### 3. Load balancing approach

To define a load balancing system for the previous distributed system, we propose an aspect (*LoadbalancingAspect*) which will grafter three code advices to the dispatcher agent in the following join cuts:

- **The diagnostic test code advice** is executed at the deployment time of the dispatcher agent. In this device, all worker agents will be asked to execute a reference task  $T_0$  in order to evaluate the computing performance and the communication latency cost of each node of the distributed system. This procedure allows establishing a baseline data for distributed load estimation of any given task.
  - **The refreshing code advice** is executed when the dispatcher agent is notified by the management system agent which signals that a new worker agent has joined or left the platform. After this event handler, this code advice will evaluate the initial performance of the new worker agent and update the load balancing parameters.
  - **The task distribution code advice** is used to implement the load balancing algorithm which determines the under loaded worker agent who will be assigned to execute the current task  $T_i$ . In this algorithm, we take into account the heterogeneity of the processing performances, the communication latency, the occupancy rate of each node and the complexity of the current task. In this model we can also use metadata of the tasks to estimate more accurately the required times to perform each task. This mechanism allows predicting the performance offered by the distributed system according to the selected computational model.
- The communication diagram of figure 3 shows how the load balancing aspect is used to balance the distributed system by the following procedure:

- At the deployment time of the dispatcher agent, the init method is executed. After that, the first code advice of the load balancing aspect is executed. In this code advice a reference task  $T_0$ , is firstly executed by the aspect in the local node of the dispatcher agent. Then the same task  $T_0$  is sent to all worker agents  $AW_i$  in order to evaluate their processing performances and their communication latency. After receiving the result  $R_0$  from all worker agents, the aspect determine the performance index corresponding to each worker agent taking into account the number of CPUs used in each node.
- When a new worker agent is registered in the multi agent platform, the agent management system (AMS) notifies the dispatcher agent about the event. At this time, the second code advice of the load balancing aspect is executed in order to evaluate the performance of the new worker agent and then update the load balancing parameters.
- When the dispatcher agent receives a new task  $T_i$  from the producer agent, the third code advice of the load balancing aspect is executed in order to determine which worker agent is the under loaded one to be assigned to perform the current task. When the task is accomplished by the designed worker agent, the result  $R_i$  is sent to the producer agent.



**Figure 3.**The task distribution model with the load balancing aspect

### 3.1. Agent worker performance index determination

The proposed load balancing procedure is performed by the load balancing aspect associated to the dispatcher agent (AD) deployed in the host node  $N_0$ . The AD agent should determine the performance index of each node of the distributed system. To do so, this agent starts by identifying the list of the  $n$  worker agents  $AW_j$  from the directory facilitator (DF) of the multi agent system to launch the diagnostic performance test. Each agent WA is asked to perform a given reference task  $T_0$  ( $x_0, y_0, \theta R_0$ ) where  $x_0$  represents the task's size,  $y_0$  represents the size of the returned result, and  $\theta R_0$  represents the execution reference duration of the task  $T_0$ . In this diagnostic test, it is important to choose a task where communication latency  $\theta L$  which depends on task size is very small and negligible compared to the computation duration  $\theta P$  which depends on the algorithm complexity of the task. Also, the computation time of this test program should be very short to achieve quickly this diagnostic test. The node performance index determination is computed by the following steps:

- **Step 1: Diagnostic test of the local node  $N_0$  :**

The reference task  $T_0$ , is firstly executed by the load balancing aspect in the local node  $N_0$ , where the dispatcher agent is deployed. This step allows us to know the reference processing duration  $\theta_0$  of the reference task  $T_0$  in the local node  $N_0$ . Since the same task must be executed by the distributed worker agents  $WA_j$ , deployed in local or remote nodes, we can measure the communication cost between the distributed agents including those deployed in the local node  $N_0$ .

- **Step 2: Diagnostic test of the node  $N_i$  :**

After the first step, the reference task  $T_0$  is sent at time  $t_0$  to all the worker agents  $AW_j$  deployed in the node  $N_j$ . Upon receipt of this data, each worker agent executes the reference task  $T_0$  and returns to the dispatcher agent, the reference result data  $R_0$  of size  $y_0$  including the processor cores count  $c_j$  and the processing duration  $\theta P_{0,j}$  elapsed during the execution of this task in the node  $N_j$ .

Upon receipt of the returned results by each node  $N_j$ , at time  $t_1(j)$ , the load balancing aspect can determine the total duration (  $\theta_{0,j} = t_1(j) - t_0$  ), necessary to perform the task  $T_0$  in each node  $N_j$  of the distributed system. The duration  $\theta_j$  includes the processing duration and the communication latency between  $N_j$  and  $N_0$ .

The communication Latency of each node  $N_j$ , can be easily reported by:

$$\theta L_{0,j} = \theta_{0,j} - \theta P_{0,j} \quad (1)$$

Taking into account the parallel processing of each node  $N_j$  using  $c_j$  cores, we can say that the number  $c_j$  of tasks  $T_0$  can be executed in parallel within the same duration  $\theta P_{0,j}$ .

$$\theta C_{0,k} = \frac{\theta P_{0,j}}{c_j} \quad (2)$$

- **Step 3: Determination of the performance index of the node  $N_j$**

The performance index of each node  $j$  can be measured using the following equation.

$$P_j = \frac{\min_{k=0}^{n-1} (\theta L_{0,k} + \theta C_{0,k})}{\theta L_{0,k} + \theta C_{0,k}} \quad (3)$$

The coefficient  $P_i$  measures the ratio of performance, taking into account the communication latency between nodes  $N_i$  and the fastest node and the number of cores of each node.  $P_i$  is always less than or equal to 1.



### 3.2. Load distribution

After this referencing phase, we suppose that the dispatcher agent has received a set of  $m$  tasks  $T_i(x_i, y_i, \theta R_i)$  where  $x_i$  represents the task size,  $y_i$  represents the size of the returned result  $R_i$ , and  $\theta R_i$  represents the execution reference duration of the task  $T_i$ .

The  $m$  tasks  $T_i$  must be distributed to  $n$  nodes of the distributed system. Each agent worker  $AW_j$  is assigned to execute simultaneously the load  $L_j$  tasks. Each task is executed in a thread.

Without using any load balancing algorithm, the dispatcher will distribute the load amount  $LO_j$  determined by the following expression:

$$LO_{i,j} = \frac{m}{n} \quad (4)$$

Using the metadata  $x_i$ ,  $y_i$  and  $\theta R_i$  of the task  $T_i$ , we can determine a theoretical estimation of the computation duration and the latency required to execute this task in a node  $N_j$ .

- *Computation time estimation:*

The computation duration of the task  $T_j$ , within the node  $N_i$  can be evaluated as in the following equation:

$$\theta P_{i,j} = \theta R_i \frac{\theta P_{0,j}}{\underset{k=0}{\overset{k=n-1}{\text{MIN}}(\theta P_{0,k})}} \quad (5)$$

- *Communication latency estimation:*

The communication latency relating to each node  $N_i$  can be determined according to the reference latency communication  $\theta L_{0,j}$  of equation (1), the input data size and the output data size of the reference task  $T_0$  and  $T_i$ .

$$\theta L_{i,j} = \theta L_{0,j} \frac{x_j + y_j}{x_0 + y_0} \quad (6)$$

- *Total processing duration estimation*

The total processing duration within each node  $N_i$  for a given data set  $D_k$  is the sum of the computation and latency durations.

$$\theta T_{i,j} = \theta P_{i,j} + \theta L_{i,j} \quad (7)$$

- *Theoretical performance factor  $P_i$  determination:*

The performance factor of each node  $j$  can be measured using the minimum of the processing durations in all nodes  $\text{MIN}(\theta P_{i,k})$  ( $k=0$  to  $n-1$ ).

$$P_j = \frac{\sum_{k=0}^{n-1} \min(\theta L_{i,k} + \theta C_{i,k})}{\theta L_{i,k} + \theta C_{i,k}} \quad (8)$$

- *Theoretical Load  $L_j$  :*

The new load  $L_i$  to apply to each node, can be calculated as in the equation (4)

$$L_j = m \cdot \frac{P_j}{\sum_{k=0}^{n-1} P_k} \quad (9)$$

- *Rounded load  $LR_j$  :*

To distribute tasks, we should use an integer value of  $L_i$ . We note this integer load  $LR_j$

$$LR_j = \text{round}(L_j) \quad (10)$$

The following difference:

$$R = m - \sum_{k=0}^{n-1} LR_k \quad (11)$$

must be distributed randomly to nodes generating a new integer loads noted  $LR2_j$

$$LR2_j = LR_j + u_j \quad (12)$$

$$\text{where } \begin{cases} u_j = 0, 1, -1 \\ \sum_{k=1}^{n-R} u_k = R \end{cases}$$

- *Total task duration required to perform the load  $L_i$  :*

Each node  $N_j$  must execute  $L_j$  tasks. The total duration of tasks in each node can be determined by:

$$\begin{aligned} \theta N_{i,j} &= L_{i,j} \left( \theta L_{i,j} + \theta C_{i,j} \right) \\ &= L_i \left( \theta L_{0,j} \frac{x_j + y_j}{x_0 + y_0} + \frac{\theta P_{i,j}}{c_j} \right) \\ &= L_i \left( \theta L_{0,j} \frac{x_j + y_j}{x_0 + y_0} + \frac{\theta R_i}{c_j} \frac{\theta P_{0,j}}{\sum_{k=0}^{n-1} \min(\theta C_{0,k})} \right) \end{aligned} \quad (13)$$

The goal is to achieve a perfect load balancing system, where this duration,  $\theta N_{i,j}$ , should be independent on the node  $N_j$ . This means that, at the same time, all the nodes should finish the execution of their jobs. So, the following difference should be close to zero.

$$\Delta \theta N_{i,j} = \theta N_{i,j}^{MAX} - \theta N_{i,j}^{MIN} \approx 0 \quad (14)$$

#### 4. Application and Results

We suppose that the dispatcher agent has received a set of  $m=100$  homogeneous tasks  $T_i(x_i, y_i, \theta R_i)$  where  $x_i$  represents the task size ( $x=4000$  KB),  $y_i$  represents the size of the returned result  $R_i$  ( $y=4000$  KB) and  $\theta R_i$  represents the execution reference duration of the task  $T_i$ . ( $\theta R_i=300$  sec)

The  $m$  tasks  $T_i$  must be distributed to  $n$  nodes ( $n=20$ ) of the distributed system. Each agent worker  $AW_j$  is assigned to execute simultaneously the load  $L_j$  tasks. Each task is executed in a thread. If we suppose that the nodes of distributed system are homogeneous, each node will process  $L_{NB} = m/20 = 5$  tasks. In our case, we assume that the distributed system is heterogeneous. So we must evaluate the performance processing and the communication latency of each node.

The reference diagnostic test, for evaluating the initial performance of the nodes of the distributed system, has been carried out by performing in a node  $N_0$  a reference task  $T_0(x_0, y_0)$ ,  $x_0=y_0=200$  KB. In table I, we present results of the performance diagnostic test using the task  $T_0$ . In this table we show, for each node, the reference processing time using one CPU core  $\theta P_{0,j}$ , the reference communication latency  $\theta L_{0,j}$ , the total reference computation time  $\theta T_{0,j}$ , the count of CPU cores  $c_j$  and the reference processing time using all the CPU cores of the node  $\theta PC_{0,j}$ .

Table 1: The performance diagnostic test results

Node	$\theta P_{0,j}$ (s)	$\theta L_{0,j}$ (s)	$\theta T_{0,j}$ (s)	$c[j]$	$\theta PC_{0,j}$ (s)
0	3,187	0,095	3,282	4	0,79675
1	4,1	0,13	4,23	2	2,05
2	3,9	0,14	4,04	3	1,3
3	4,2	0,132	4,332	4	1,05
4	3,1	0,21	3,31	4	0,775
5	3,3	0,252	3,552	3	1,1
6	4,79	0,22	5,01	2	2,395
7	3,28	0,19	3,47	3	1,093333333
8	4,32	0,19	4,51	3	1,44
9	4,81	0,226	5,036	5	0,962
10	4,008	0,115	4,123	3	1,336
11	4,12	0,12	4,24	3	1,373333333
12	3,955	0,115	4,07	3	1,318333333
13	5,114	0,118	5,232	3	1,704666667
14	3,187	0,123	3,31	2	1,5935
15	3,782	0,098	3,88	2	1,891
16	4,84	0,27	5,11	3	1,613333333
17	3,19	0,13	3,32	4	0,7975
18	4,07	0,17	4,24	4	1,0175
19	3,88	0,206	4,086	3	1,293333333

In table II, we present computed and experiment results obtained for the task  $T_i$ . In this table we show, for each node, the computing time  $\Theta P_j$ , the latency communication  $\Theta L_j$ , the performance index  $P_j$ , the loads  $L_j$ , the rounded loads  $LAr_j$ , the theoretical processing time of one task  $\Theta T_j$ , the theoretical processing time of  $L_j$  tasks  $\Theta N_j$ , the experiment processing time of  $L_j$  tasks  $\Theta R_j$ , the experiment processing time of  $LAr_j$  tasks if the load balancing aspect is not applied  $\Theta N_0[j]$ .

Table 2: Theoretical and experiment results obtained for the task  $T_i$

Node	$\Theta P[j]$ (s)	$\Theta L[j]$ (s)	$P[j]$	$L[j]$	$LAr[j]$	$\Theta T[j]$ (s)	$\Theta N[j]$ (s)	$\Theta R[j]$ (s)	$\Theta N_0[j]$ (s)
0	308,419	1,900	0,980	7,692	7	310,319	2387,117	2173	1551,597
1	396,774	2,600	0,382	2,998	2	399,374	2387,117	1591	3980,742
2	377,419	2,800	0,601	4,717	4	380,219	2387,117	2034	2530,129
3	406,452	2,640	0,744	5,835	6	409,092	2387,117	2457	2045,458
4	300,000	4,200	1,000	7,847	8	304,200	2387,117	2437	1521,000
5	319,355	5,040	0,706	5,541	6	324,395	2387,117	2583	2154,232
6	463,548	4,400	0,327	2,563	3	467,948	2387,117	2792	4657,484
7	317,419	3,800	0,712	5,590	6	321,219	2387,117	2564	2135,129
8	418,065	3,800	0,542	4,253	4	421,865	2387,117	2251	2806,097
9	465,484	4,520	0,807	6,333	6	470,004	2387,117	2263	1884,535
10	387,871	2,300	0,586	4,595	5	390,171	2387,117	2591	2597,306
11	398,710	2,400	0,570	4,470	4	401,110	2387,117	2135	2670,065
12	382,742	2,300	0,593	4,657	5	385,042	2387,117	2566	2563,113
13	494,903	2,360	0,459	3,605	4	497,263	2387,117	2647	3311,155
14	308,419	2,460	0,491	3,855	4	310,879	2387,117	2471	3096,494
15	366,000	1,960	0,414	3,252	3	367,960	2387,117	2207	3669,800
16	468,387	5,400	0,483	3,790	4	473,787	2387,117	2518	3149,581
17	308,710	2,600	0,977	7,668	8	311,310	2387,117	2488	1556,548
18	393,871	3,400	0,766	6,009	6	397,271	2387,117	2387	1986,355
19	375,484	4,120	0,603	4,729	5	379,604	2387,117	2534	2523,826

The curves of figures 4 and figure 5 represent the performance index of each node and the load  $LAr_j$  of tasks attributed to each node respectively.

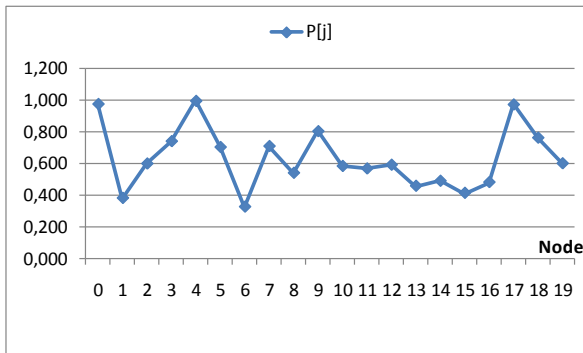


Figure 4. Performance index of each node

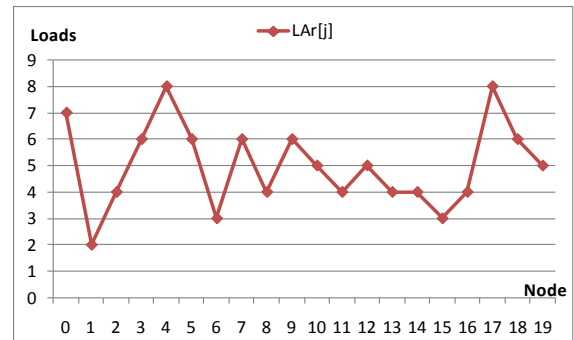


Figure 5. The number of tasks attributed to each node

The curve of the figure 6 represents the computation time  $\Theta P0_j$ , the communication latency  $\Theta L0_j$  and the computation time  $\Theta PC0_j$  obtained in the reference diagnostic test using the task  $T_0$ . we notice that the communication latency is negligible compared to the computation duration and the real computation performances of each node represented by  $\Theta PC0_j$  and computed taking into account the number of CPU's cores.

In the curve of the figure 7, we show the execution time of the amount  $L_j$  tasks  $T_i$  affected to each worker agent. We can see that, using our mathematical model, we obtain a perfect load balancing stat of the distributed system. The execution time within each node is the same:  $\text{MAX}(\Theta N_j) = \Theta N_j = 2387,117 \text{ sec}$ . Without applying the load balancing aspect, we notice that the system is not balanced and the execution duration of the attributed loads is different in each node. In this case, the slowest node of the distributed system is the node 6 where the execution time  $\text{Max}(\Theta N0_j) = 4657,484 \text{ sec}$  and the fastest node to accomplish the attributed loads is the node 4 where the execution time  $\text{MIN}(\Theta N0_j) = 1521 \text{ sec}$ . we can say that the saved time when we use the load balancing aspect is  $\Theta S = \text{Max}(\Theta N0_j) - \text{MAX}(\Theta N_j) = 2270,367 \text{ sec}$ . The theoretical performance gain is evaluated at:

$$\mu 1 = \frac{\text{MAX}(\Theta N0_j) - \text{MAX}(\Theta N_j)}{\text{MAX}(\Theta N_j)} = 95,11\%$$

In the experimental case, where we use the rounded loads  $LA r_j$ ,  $\text{MAX}(\Theta R_j) = 2792 \text{ sec}$ . This value is obtained in the node 6. The saved time is evaluated at  $\Theta S = \text{Max}(\Theta N0_j) - \text{MAX}(\Theta R_j) = 1864,994 \text{ sec}$  and the performance gain in this case is:

$$\mu 2 = \frac{\text{MAX}(\Theta N0_j) - \text{MAX}(\Theta R_j)}{\text{MAX}(\Theta R_j)} = 66,78\%$$

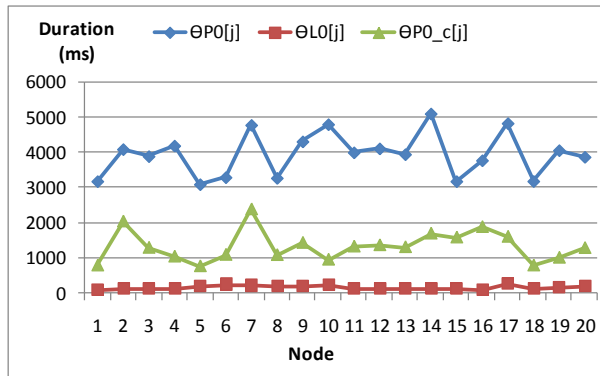


Figure 6. Execution times in each node

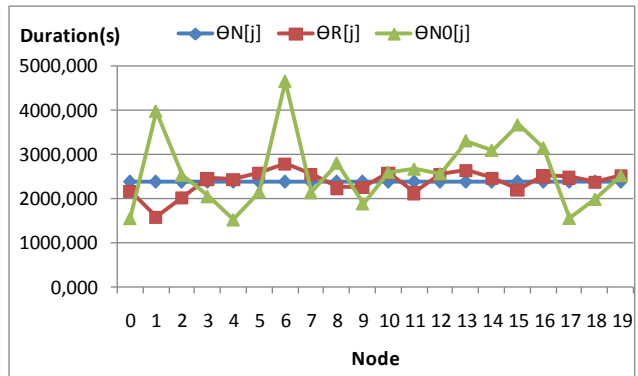


Figure 7. Execution times in each node

## 5. Conclusion

In this paper, we have presented a new load balancing system based on aspect oriented approach for multi agents distributed systems. The proposed model is based on using a dispatcher agent assigned to dynamically distribute tasks received in random requests coming from agent producers. We suppose that tasks are assigned to be executed by workers agents deployed over the nodes of the distributed system. To make the initial mechanism of task distribution balanced, we propose a separated aspect where we propose a load balancing procedure assigned to be affected to the dispatcher by an aspect weaving in compilation or run times. The load balancing aspect increases the performances of the dispatcher agent using three code advices in different join points which are the diagnostic test code advice the refreshing code advice and the task distribution code advice used to implement the load balancing algorithm. This algorithm determines the under loaded worker agent which will be assigned to execute the current task  $T_i$ . In this algorithm, we take into account the heterogeneity of the processing performances, the communication latency, the occupancy rate of each node and the complexity of the current task. In this model we can also use metadata of the tasks to estimate more accurately the required times to perform each task. This mechanism allows predicting the performance offered by the distributed system according to the selected computational model. To describe the proposed approach, we have used some communication diagrams, the established mathematical model and an application example. The obtained results prove the effectiveness of the proposed model. As a perspective, we will compare these results to other similar works in the literature.

## References

- [1] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of High Performance Computing Applications*, **15** (2001), no. 3, 200–222.  
<http://dx.doi.org/10.1177/109434200101500302>
- [2] Mohamed Youssfi, Omar Bouattane, Mohammed Ouadi Bensalah, A Parallel Computational Model Based on Mobile Agents for High Performance Computing, *Contemporary Engineering Sciences*, **8** (2015), no. 15, 677- 698.  
<http://dx.doi.org/10.12988/ces.2015.55153>
- [3] Omar Bouattane, Bouchaib Cherradi, Mohamed Youssfi and M.O. Bensalah Parallel c-means algorithm for image segmentation on a reconfigurable mesh computer, *Parallel computing*, **37** (2011), 230-243.  
<http://dx.doi.org/10.1016/j.parco.2011.03.001>

- [4] Dongsheng Li, Jiannong Cao, Xicheng Lu, K. Chen, Efficient Range Query Processing in Peer-to-Peer Systems, *IEEE Transactions on Knowledge and Data Engineering*, **21** (2009), no. 1, 78-91.  
<http://dx.doi.org/10.1109/tkde.2008.99>
- [5] D.R. Karger and M. Ruhl, Simple Efficient Load-Balancing Algorithms for Peer-to-Peer Systems, *Theory of Computing Systems*, **39** (2006), 787-804.  
<http://dx.doi.org/10.1007/s00224-006-1246-6>
- [6] Ioannis Konstantinou, Dimitrios Tsoumakos, Nectarios Koziris, Fast and Cost-Effective Online Load-Balancing in Distributed Range-Queryable Systems, *IEEE Transactions on Parallel and Distributed Systems*, **22** (2011), no. 8, 1350-1364. <http://dx.doi.org/10.1109/tpds.2010.200>
- [7] Mohamed Youssfi, Omar Bouattane, Mohammed Ouadi Bensalah, Efficient load balancing algorithm for distributed systems using mobile agents, *Advanced Studies in Theoretical Physics*, **9** (2015), no. 5, 245-253.  
<http://dx.doi.org/10.12988/astp.2015.5110>
- [8] G. Cybenko, Dynamic Load balancing for distributed memory multiprocessors, *Journal of Parallel and Distributed Computing*, **7** (1989), 279-301. [http://dx.doi.org/10.1016/0743-7315\(89\)90021-x](http://dx.doi.org/10.1016/0743-7315(89)90021-x)
- [9] Rupali Bhardwaj, V.S. Dixit, Anil Kr.Upadhyay, A Propound Method for Agent Based Dynamic Load Balancing Algorithm for Heterogeneous P2P Systems, *2009 International Conference on Intelligent Agent and Multi-Agent Systems*, 2009, 1-4.  
<http://dx.doi.org/10.1109/iama.2009.5228060>
- [10] F.M. auf der Heide, B. Oesterdiekhoff, and R. Wanka, Strongly adaptive token distribution, *Algorithmica*, **15** (1996), 413-427.  
<http://dx.doi.org/10.1007/s004539900023>
- [11] Ioannis Konstantinou, Dimitrios Tsoumakos, NectariosKoziris, Fast and Cost-Effective Online Load-Balancing in Distributed Range Queryable Systems, *IEEE Transactions on Parallel and Distributed Systems*, **22** (2011), no. 8, 1350-1364. <http://dx.doi.org/10.1109/tpds.2010.200>
- [12] Jie Li, Hisao Kameda, Load Balancing Problems for Multiclass Jobs in Distributed/Parallel Computer Systems, *IEEE Transactions on Computers*, **47** (1998), no. 3, 322-332. <http://dx.doi.org/10.1109/12.660168>
- [13] 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.*, **62** (2002), no. 12, 1729-1746.  
[http://dx.doi.org/10.1016/s0743-7315\(02\)00006-0](http://dx.doi.org/10.1016/s0743-7315(02)00006-0)

- [14] Y.F. Hu, R.J. Blake, An Improved diffusion algorithm for dynamic load balancing, *Parallel Computing*, **25** (1999), 417-444.  
[http://dx.doi.org/10.1016/s0167-8191\(99\)00002-2](http://dx.doi.org/10.1016/s0167-8191(99)00002-2)
- [15] E. Luque, A. Ripoll, A. Cortes and T. Margalef, A Distributed Diffusion method for dynamic load balancing on parallel computers, *IEEE Parallel and Distributed Processing, 1995. Proceedings. Euromicro Workshop, 1995*, 43-50. <http://dx.doi.org/10.1109/empdp.1995.389156>
- [16] Tina A. Murphy and John G. Vaughan, On the Relative Performance of Diffusion and Dimension Exchange Load Balancing in Hypercubes, *Procc. of the Fifth Euromicro Workshop on Parallel and Distributed Processing, PDP'97, 1997*, 29-34.
- [17] P. Berenbrink, T. Friedetzky, and Z. Hu, A new analytical method for parallel, diffusion-type load balancing, *J. Parallel Distrib. Comput.*, **69** (2009), no. 1, 54–61. <http://dx.doi.org/10.1016/j.jpdc.2008.05.005>
- [18] F. Cedo, A. Cortes, A. Ripoll, M. A. Senar, and E. Luque. The convergence of realistic distributed loadbalancing algorithms, *Theor. Comp. Sys.*, **41** (2007), no. 4, 609– 618. <http://dx.doi.org/10.1007/s00224-006-1214-1>
- [19] Raghu Subramain, Issac D. Scherson, An Analysis of Diffusive Load-Balancing, *Proceedings of 6th ACM Symposium on Parallel Algorithms and Architectures*, 1994, 220-225.  
<http://dx.doi.org/10.1145/181014.181361>
- [20] Yihua Wu, Jian Cao, Minglu Li, Private Cloud System Based on BOINC with Support for Parallel and Distributed Simulation, *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011, 1172-1178. <http://dx.doi.org/10.1109/dasc.2011.190>
- [21] K.W. Ross and D.D. Yao, Optimal Load Balancing and Scheduling in a Distributed Computer System, *Journal of the ACM*, **38** (1991), no. 3, 676-690. <http://dx.doi.org/10.1145/116825.116847>
- [22] T. Friedrich and T. Sauerwald, Near-perfect load balancing by randomized rounding, *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, 2009, 121-130. <http://dx.doi.org/10.1145/1536414.1536433>
- [23] Marc H. Willebeek-LeMair, Anthony P. Reeves, Strategies for Dynamic Load Balancing on Highly Parallel Computers, *IEEE Transaction on Parallel and Distributed Systems*, **4** (1993), no 9, 979-993.  
<http://dx.doi.org/10.1109/71.243526>



- [24] Y. Qiao, G. von Bochmann, A diffusive load balancing scheme for clustered peer-to-peer systems, *Proceedings of the 2009 15th ICPADS. IEEE Computer Society*, 2009, 842–847.  
<http://dx.doi.org/10.1109/icpads.2009.119>
- [25] S. Muthukrishnan, B. Ghosh, and M. Schultz, First and second-order diffusive methods for rapid, coarse, distributed load balancing, *Theory of Computing Systems*, **31** (1998), no. 4, 331–354.  
<http://dx.doi.org/10.1007/s002240000092>
- [26] Liu, J., Jin, X. and Wang, Y. Agent-Based Load Balancing on Homogeneous Mini-grids: Macroscopic Modeling and Characterization, *IEEE Transactions on Parallel and Distributed Systems*, **16** (2005), 586-598.  
<http://dx.doi.org/10.1109/tpds.2005.76>
- [27] D. Acker, S. Kulkarni, A Dynamic Load Dispersion Algorithm for Load Balancing in a Heterogeneous Grid System, *IEEE Sarnoff Symposium*, (2007), 1- 5. <http://dx.doi.org/10.1109/sarnof.2007.4567375>
- [28] F. Bellifemine, A. Poggi, G. Rimassa, JADE -- A FIPA-compliant agent framework, CSELT internal technical report. Part of this report has been also published in Proceedings of PAAM'99, (1999), 97-108.
- [29] C. Castelfranchi, Y. Lespérance, Developing Multi-agent Systems with JADE Intelligent Agents, In *Intelligent Agents VII Agent Theories Architectures and Languages*, Vol. 1986, (2001).
- [30] Erik Hilsdale, Jim Hugunin, Advice weaving in AspectJ, *Proceedings of the 3rd international conference on Aspect-oriented software development*, 2004, 26-35, March 22-24. <http://dx.doi.org/10.1145/976270.976276>
- [31] M. Lippert, C.V. Lopes, A study on exception detection and handling using aspect-oriented programming, *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, 418,427.  
<http://dx.doi.org/10.1109/icse.2000.870432>

**Received: December 28, 2015; Published: February 5, 2016**