MLBLM: A Multi-level Load Balancing Mechanism in Agent-Based Grid

Mohsen Amini Salehi[1], Hossain Deldari[2], and Bahare Mokarram Dorri[3]

[1] Department of Software Engineering, Faculty of Engineering, Islamic Azad University, Mashhad Branch, Iran
[2] Department of Software Engineering, Ferdowsi University, Mashhad, Iran
[3] Management and Planning Organisation of Khorasan, Mashhad, Iran
Amini@mshdiau.ac.ir, hd@ferdowsi.um.ac.ir, mokarram@mpo-kh.ir

Abstract. A computational grid is a widespread computing environment that provides huge computational power for large-scale distributed applications. Load balancing, has a considerable effect on the grid middleware performance. Current load balancing methods cannot satisfy all necessities for the grid. In this paper, a Multi-level Load Balancing Method (MLBM) is proposed. Cooperation among different levels in this method, removes disadvantages of each level, while satisfy most of load balancing requirements needed. Simulation results indicate that this new mechanism surpasses its predecessors in increasing efficiency and decreasing communication overhead.

1 Introduction

A computational grid is a hardware and software infrastructure that provides consistent, pervasive and inexpensive access to high end computational capacity. An ideal grid middleware should provide access to all the available resources seamlessly and fairly [1].

ARMS is an agent-based resource manger for grid computing which is aimed at provisioning scalability and adaptability [1]. In this system, agents cooperate with each other to achieve resource discovery. Each agent organizes all service information of a resource into Agent Capability Tables (ACTs). The agents are equipped with a performance prediction toolkit called PACE [1], [2] to predict available efficiency of resources. Experiments testify a high level of precision attained through PACE.

Considering the largeness, dynamic resources, and other specifications of the grid, it is impossible to utilize resources in equilibrium, unless using efficient load balancing methods. However, lack of a well-organized load balancing method is a crucial problem in most of grid resource managers, like ARMS.

Taking into account the ARMS specifications and the importance of load balancing, in this work, we attempt to propose a multi layer load balancing mechanism for ARMS.

Load balancing methods are designed essentially to spread the load on resources equally and maximize their utilization while minimizing the total task execution time [3]. Recently, some methods have been suggested for load balancing in the grid [1], [4], [5]. Heuristic approaches are applied in most of these methods.

S. Chaudhuri et al. (Eds.): ICDCN 2006, LNCS 4308, pp. 157 – 162, 2006.

J. Cao implemented a load balancing method in ARMS [1]. As stated in [1], the efficiency of the mechanism highly depends on the number of cooperating ants *(n)* as well as their step count *(m)* which is defined by the grid user itself.

Some load balancing methods, like QLBVR [6], are periodical. It balances the extra load among neighboring nodes according to their average queue length and request arrival rate. This method uses *virtual routing* [6] method for checking the balancing profitability. Virtual routing changes the load balancing difficulty to an optimal routing problem by adding a virtual node in the network system.

In our proposed method, which is provided in the next section, we intend to use the advantages of both attitudes. Furthermore, there is an effective load balancing method in ARMS, which provides an optimal scheduling within a node [2]. We call this method a '*local-level*' load-balancing and we use it as the first level of load balancing in our new multi-layer approach, MLBM.

The rest of the paper is organized as follows: Section 2 contains a survey on current load balancing methods. In Section 3, different levels of MLBM method are described. Performance metrics and simulation results are included in Section 4. At last, we will present the conclusion as well as the relevant future works.

2 Proposed Method

In this section, firstly, a new load balancing method based on ant colony heuristic is proposed, and then a complementary method, which tries to compensate its defects, is suggested. Coupling these two methods with *local-level* will construct MLBM.

In this paper, the number of waiting jobs is considered as a criterion for measuring load in a node.

2.1 Grid-Level Load Balancing

In this level, an echo system of intelligent ants is suggested. Interactions between these ants will result in load balancing throughout the grid. Here, echo system means that the ants are created on demand to achieve load balancing. They may bear offspring or they commit suicide according to their environmental conditions. Every ant in the new mechanism hops *'m'* steps and then balances *'k'* overloaded nodes with *'k'* underloaded. In the next subsections, we will describe the grid-level method.

2.1.1 Creating, Moving and Deciding of Ants

If a node understands that it is overloaded, it can create a new ant with a few steps to balance the load quickly. A memory space, which is divided into an underloaded list *(Min-List)* and an overloaded list *(Max-List)*, is allocated to each ant in which the ant records specifications of the overloaded and underloaded nodes while wanders.

After entering a node, the ant should determine state of the node, i.e. overloaded, underloaded or equilibrium, using its acquired knowledge from the environment. As the state of the node is determined relative to the system conditions, decision making is performed adaptively by applying adaptive fuzzy logic. To make a decision, the ant deploys the node's current workload and its (i.e. the ant's) remained steps as two inputs to the fuzzy inference system. Then, the ant determines the state of the node.

Therefore, If the result is *"overloaded"* or *"underloaded"*, the node specifications must be added to the ant's *max-list* or *min-list*. Subsequently, the corresponding counter for *Max*, *Min*, or *Avg* increases by one.

In special circumstances, especially when an ant's life span is long while the ant is continuing its wandering, its memory may get full, but it still encounters nodes which are overloaded or underloaded. In this situation, if a node load is overloaded, the ant bears a new one with predefined steps. Here, adaptability translates into increasing the number of the ants automatically, whenever there are many overloaded nodes.

2.1.2 Load Balancing, Starting New Itineration

When the ant's hops end, it must start the balancing operation between its overloaded (*Max*) and underloaded (*Min*) elements gathered. After load balancing, the ant must reinitiate to begin a new itineration. One of the fields that must be initiated is the ant's step counts. However, the ant's step counts *(m)* must be relative to system conditions [1]. Therefore, if most of the nodes visited were underloaded or in equilibrium, the ant should prolong its wandering steps, i.e. decrease the load balancing frequency and vice versa. Doing this requires the ant's knowledge about the environment. Adaptive fuzzy logic is again used in determining the next itineration step counts. The controller determines the next step counts (*NextS*) based on the number of overloaded, underloaded and equilibrium nodes visited, along with the step counts during the last itineration (*LastS*). Actually, the former indicates recent condition of the environment, while later reports lifetime history of the ant. This fuzzy system can be stated as a relation as follows:

$$\begin{aligned} &R_A : MaxCnt < l,m,h > * MinCnt < l,m,h > * AvgCnt < l,m,h > \\ &* LastS < tl,l,m,h,th > \rightarrow NextS < tl,l,m,h,th,Dead > \end{aligned} \tag{1}$$

If an ant's step counts extend to extreme values, its effect tends to be zero. Based on this premise, one can conclude that an ant with too long step counts does not have any influence on the system balance. In this circumstance, the ant must commit suicide i.e. *NextS* is fired in the *"Dead"* membership function.

2.2 Neighbor-Level Load Balancing

As mentioned before, we use neighbor-level load balancing as the second layer in MLBM. The algorithm of this level works as follows:

In ARMS, agents use PACE to obtain their resource capabilities information. Agents periodically exchange this information with their neighbors. An agent advertises its load information only among its neighbors. Load characteristics of the nodes could be attached to this exchanging information. Moreover, the receiver agent can estimate the time gap between sending and receiving the information as the transmission cost. The agents in the system exchange their status information at periodic interval of time T_s. We opportunistically append our favorite fields to them.

Based on the load information received in the last T_s, the agents estimate the current load of their neighbors in each interval. Then, the agent computes the average load on its neighboring agents. An agent calls itself "*overloaded*" if its load is greater

than the average load of its neighbors. Neighbors, whose estimated load is less than the average, form an *active set*. However, because of the transmission cost, sending the agent load to all of the active set members might not be profitable [7]. We compass the problem of profitability using virtual routing [6].

2.3 MLBM: Multi-level Load Balancing Mechanism

Each of the two proposed methods has some defects. Neighbor-level method has a limited vision and grid-level method imposes too much communication overhead and is not fair. Combining these two methods with the *local-level*, would satisfy most of requirements for an ideal load balancing method. Now, MLBM works as follows:

Agents, periodically, exchange their state information in ARMS. Load information is attached to this exchanging data. Each agent uses *local-level* method to make its resources balance. Moreover, each overloaded agent uses *neighbor-level* method, to balance its load with adjacent neighbors. At the same time, some ants may pass through that agent and choose it for further balancing. However, if a node is overloaded, for several periods of time, and it has not been visited during this time, then the node itself creates a new ant to balance its load throughout a wider area.

Consider that in *neighbor-level* method, scattering radius is limited; however this flaw was compensated using ants. On the other side, using *neighbor-level*, the load inequality decreases. This causes fewer ants and less communication overhead. Moreover, as each node uses *neighbor-level* method, even if it is not visited by any ant, it can achieve load balancing. Thus the mechanism is fair.

3 Performance Evaluation

There are a number of performance metrics used to describe grid scheduling systems which are investigated. Let P be the number of agents of ARMS system and W_{pk} (p=1, 2… P) be the workload of the agent p at period k. The average workload is:

$$\overline{W_k} = \frac{\sum_{p=1}^{P} W_{pk}}{P} \tag{2}$$

The mean square deviation of W_{pk}, which characterizes the load balancing level of the system, is defined as (3). Let T_k be the total time spent in all agents to achieve a load balancing level L_k. Then, load balancing efficiency e_k , is calculated according to (4).

$$L_k = \sqrt{\frac{\sum_{p=1}^{P} (\overline{W_k} - W_{pk})^2}{P}} \tag{3}$$

$$e_k = \frac{L_0 - L_k}{T_k} \tag{4}$$

In this work, Agent system, Workload, and Resources are modeled as follows:

- *Agents*. Agents are mapped to a square grid. This simplification has been done in similar works [1], [4], and [5]. All of experiments described later include 400 agents.
- *Workload*. A workload value and corresponding distribution are used to characterize the system workload. The value is generated randomly in each agent.
- *Resources*. Resources are defined in the same way as workload.

4 Simulation Results

First experiment involves total network connections needed. As shown in Fig.1, total communication needed (*C*), in MLBM, is drastically less than the conditions only grid-level or seminal method [1] is used. It can be seen that the communication count goes flat in the last seconds, when the load balancing frequency decreases.

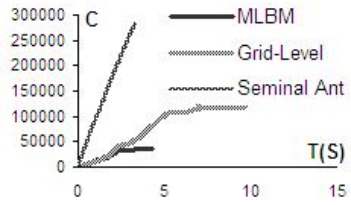

Fig. 1. Comparing communications needed (*C*) in MLBM, Grid-level, and Seminal method during the time (*T(s)*)

In second experiment, convergence speed is compared between the three methods.

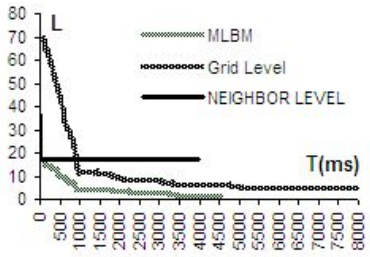

Fig. 2. Convergence speed in MLBM, Grid-level and Neighbor-level methods

For the sake of comparison, we examined balancing level (*L*) achieved during the time. The results are illustrated in Fig.2. As stated before, slow convergence speed was a defect in neighbor-level method. However, Fig.2 explains that the combination of the two methods cope the disadvantage.

In the last experiment, system efficiency is discussed. Efficiency (*e*) is calculated for MLBM, grid-level, and seminal ant method [1] during the time (*T*). Fig.3 proves that MLBM has the best efficiency between others.

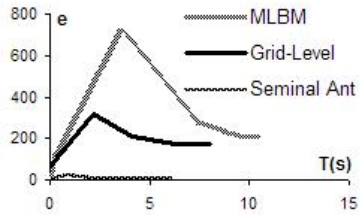

Fig. 3. Comparing efficiency (*e*) between MLBM, grid-level, seminal method in time (*T*)

5 Conclusion and Future Works

In this research, we proposed a multi-level load balancing method (MLBM) for grid environment; overloaded nodes get balances through these layers. In the first layer, which is '*node-level*', an efficient scheduler tries to use node's resources equally. The second level, which is called '*neighbor-level*', periodically scatters the extra load of overloaded nodes to a limited domain. The third level, which is '*grid-level*', is a colony of intelligent ants which spread the regional extra load throughout the grid.

Cooperation of these layers in a multi-layer framework (MLBM) alleviates their disadvantages and, as exhibited in the paper, results in better efficiency.

In our future works, we plan to prove MLBM mathematically, promoting ant's intelligence and adaptation as well as adding billing contracts between resources as they exchange customer loads and overcome security considerations.

References

1. J. Cao: Self-Organizing Agents for Grid Load Balancing, Proc. 5th IEEE/ACM Int. Workshop on Grid Computing.
2. J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd: Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling, in Proc. of 17th IEEE Int. Parallel and Distributed Processing Symp. Nice, (2003), 218-224.
3. A. Y. Zomaya and Y. The: Observations on using genetic algorithms for dynamic load-balancing, IEEE Trans. on Parallel and Distributed Systems, (2001), 899-911.
4. M. Amini, H. Deldari: Grid Load Balancing Using an Echo System of Ants, in Proc. Of 24th IASTED Int. Cnf, Innsbruck, (2006) 47–52.
5. M. Amini, H. Deldari: A Novel Load balancing Method in an Agent-based Grid, in Proc. Of IEEE Int. Cnf on Computing and Informatics, Kuala Lumpur, (2006).
6. Z. Zeng and B.Veeravalli: Rate-Based and Queue-Based Dynamic Load Balancing Algorithms in Distributed Systems, Proc. of the 10th Int. Cnf. on Parallel and Distributed Systems, (2004), 156-163.
7. S. Dhaka1, B. S. Paskaleva, M. Hayat, E. Schamiloglu, C. T. Abdallah: Dynamical Discrete-Time Load Balancing in Distributed Systems in the presence of Time Delays, in Proc. 42nd IEEE Decision and Control Vol.5, (2003), 5128-5134.