Shahram JAMALI [1], Shiva RAZZAGHZADEH[2]

Department of Computer Engineering, University of Mohaghegh Ardabili, Ardabil, Iran(1), Young Researchers Club, Ardabil Branch, Islamic Azad University, Ardabil, Iran (2)

GLBAI: Global Load Balancing Using Agents' Identifiers in Grid Environment

Abstract. *In a grid environment the resource management, task scheduling and also load balancing are essential functionalities provided by software infrastructure. Recently, intelligent agent technology has provided an adaptive and scalable framework for management and scheduling of dynamic resources in the grid environment. In this approach an agent is a representative of a grid resource that is supported by a software system. This paper presents a new agent-based method, called GLBAI, for the improvement of global load balancing. While previous works assume that all existing agents in the environment have identical capabilities, this paper considers different capabilities for different agents. It proposes an agent identifier that indicates the agent's capabilities. The agents cooperate with each other to balance workload using a service advertisement and discovery mechanism in which agent's capability information are passed in the form of the identifier. Simulation results show that GLBAI reduces application execution time, maximizes resource utilization and offers more efficient load balancing service.*

Streszczenie. *W artykule przedstawiono nową metodę agentową o nazwie GLBAI, do zarządzania zasobami energii w sieci. Metoda ma na celu ulepszenie zbalansowania globalnego obciążenia sieci, poprzez uszeregowanie różnych agentów pod względem ich możliwości zasobowych i nadanie odpowiedniego identyfikatora. Agenci współpracują między sobą w celu zbalansowania obciążenia oraz wykorzystania informacji zawartych w identyfikatorach. Badania symulacyjne wskazują, że system GLBAI redukuje czas wykonania programu, maksymalizuje wykorzystanie zasobów oraz oferuje lepszy zbalansowanie obciążenia. (**GLBAI dla sieci energetycznej – system balansowania globalnym obciążeniem sieci energetycznej, wykorzystujący identyfikatory agentów**)*

Keywords: Grid, load balancing, agents, task allocation, agent identifier.
Słowa kluczowe: sieć, balansowanie obciążeniem, agenci, alokacja zadań, identyfikator agenta.

Introduction

Grid computation is a kind of distributed calculations that constructs a bridge not only for geographical distances but also among organizations, different machine architecture and software. It makes possible for everyone who is connected to grid, the indefinite calculation power, the possibility of cooperation with others and access to different kinds of information. The main viewpoint of grid is to establish dynamic virtual organizations via resources sharing in a coordinated and secure way among users, universities and organizations in order to form a huge virtual computational/communicational bed so that it can be used for solving complicated problems which need a huge amount of data processing. The existence of heterogeneous resources which are geographically distributed and also the abundant number of these resources cause the resource management and load balancing to be more essential and important. The aim is to prevent some resources to be overloaded, while some others are low utilized.

Based on the concentration degree, the load scheduling algorithms can be divided into two types: centralized and decentralized. In centralized algorithms there is a site which acts as a central controller. Its task is monitoring the task scheduling. Whenever the task allocation cost is low, this algorithm is benefit, but whenever the size of system increases, the bottleneck problem is posed. In load balancing, based on decentralized scheduling, all sites share in task allocation. This kind of algorithm has high and better fault tolerance [1, 2, 3]. In another classification, we can consider the load balancing algorithms in the forms of static, dynamic and hybrid. The static policies often use the FCFS (first-come-first-served) algorithm and devote the tasks to the suitable resources based on simple information from system. On the other hand, dynamic policies often utilize genetic algorithm and they perform task allocation based on the present state of the system. Finally the hybrid algorithms utilize the combination of dynamic and static policies switch to each of them in an appropriate time [4, 5]. Recently, intelligent agent technology has found applications in management and scheduling of dynamic resources in the grid environment [6]. In this approach an agent is a representative of a grid resource that is supported by a software system. In this paper we presents a new agent-based method, called GLBAI, for the improvement of global load balancing. While Cao and her coworkers assume in [6] that all existing agents in the environment have identical capabilities, this paper considers different capabilities for different agents. It proposes an agent identifier that indicates the agent's capabilities. The agents cooperate with each other to balance workload using a service advertisement and discovery mechanism in which agent's capability information are passed in the form of the identifier. GLBAI decreases the number of searching steps to find effective agents by the use of agent's identifier and it also prevents from useless request sending to ineffective agents. The result of simulation shows that the GLBAI decreases the required time for task allocation and service discovery. It improves average utilization of the grid resources which in turn leads to decreased task execution time. Load balancing level offered by this approach is more than the basic approach proposed by Cao et al. in [6] which is called Cao algorithm in this paper.

The rest of the paper is organized as follows. Section 2 presents a fast review over the recent works around task scheduling and load balancing issues in the grid environment. In section 3, we bring our proposed approach describing its details. Section 4 compares the proposed mechanism with Cao algorithm through a simulative study by using Gridsim [7] simulator and finally section 5 concludes the paper.

Related works

Load balancing algorithms can be divided into two parts: the local and global [6]. Whenever the number of tasks entering to the environment is low, the FCFS algorithm is usually employed to locally schedule the tasks. On the other hand, if the number of resources increases and the environment is vast and in a large scale, the global load balancing algorithm will be suitable. This policy represents a suitable function by service advertisement and discovery too [6, 8, 9]. Another study done in this regard is method which is based on branch and bound. This algorithm is

based on tree model and proposes a dynamic load balancing policy. In this case after numbering of the nodes, each node selects an interval containing the number of node's children for itself. Then some of the tree children are put aside by the use of some formulas and pruning and finally the load balancing is achieved. This method is repeated up to getting the optimum response. This is the only shortcoming of this strategy [10]. Some of the hybrid methods also have been proposed for making load balancing. One of these methods named static which collects the information of each node via agents then it delivers the information to the dispatcher. In the next step, the dispatcher itself finds suitable and effective node by doing a set of calculations then it allocates the task to that node. The other method namely dynamic one, transfers the information of each node to dispatcher via agent whenever the nodes state change. Then the dispatcher selects the effective node by comparison of this information with previous one [5]. The only shortcoming of this method is that, switching between the dynamic and static state increases the overhead. Among the other studies done concerning with load balancing is using ant colony. In this method ants move forward randomly and wonderingly and they also interact with each other in their way. Whenever the ants see their children in their way and at the same time, when they find the environment imbalanced, they ignore their own children which interference in these circumstances. Then they commit suicide in order to make the load balancing. In this method the ants move with *m* pace. At the end of *m* pace, the load balancing is made [11]. Another important load balancing also executes its algorithm based on intelligent agents. In this method each agent is sign of a local resource. In upper steps the agents unite to make global load balancing. This method uses FCFS algorithm (whenever the number of tasks is low) and genetic algorithm (whenever the number of tasks is high) to make local load balancing. These algorithms represent suitable solutions for dynamic scheduling but in the vast environments, they are not capable. Therefore; in this condition, the global load balancing is used. It uses the service advertisement and discovery method too. This is the case that each agent makes ACT (agent capability table) for itself. This table represents the necessary information for global load balancing establishing. The main advantage of this method is that it doesn't suffer from bottleneck [6]. Scheduling algorithms in load balancing can be in the form of clairvoyant and non-clairvoyant [12]. Clairvoyant algorithm has information about job characteristics such as time servicing etc. This algorithm devotes the given job to the appropriate node by taking into consideration of these characteristics, but non-clairvoyant one has no information about the jobs therefore this algorithm doesn't contain a regular method to allocate the jobs to the resources.

The GLBAI Method

In this section we describe our proposed mechanism. As the first step, we introduce the environment in which we establish the proposed algorithm. This work adopts the environment assumed in [6] but tries to make it even more realistic to better reflect the heterogeneous nature of the grid environment. As shown in Fig. 1, the environment employed in [6] contains a broker which acts as a head of the group.

It takes over supervising the whole collection of agents. The next member is a coordinator which itself is a kind of agent and conducts the sub-hierarchy. This part delivers the required information for making decision to the agents. Finally the agents act as tree leaves and are the first service providers. As Fig. 1 shows, this is multi-agent environment in which the connections between these agents are feasible via communication layer, coordination layer and local management layer. These layers are also the structure of each agent. In this figure, the position of broker, agent and coordinator in hierarchy is different, but their function and capability are assumed to be identical. Although agents have different names in this tree hierarchy, all of them are a same kind of agents. They are ready to give service according to user's request. All of the agents are equipped to PACE (performance application capability engine) and this makes it possible for them to be able to predict the request execution speed and time of programs execution too because of its searches in the mentioned paper.

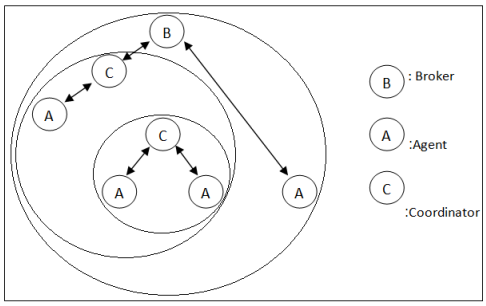

Fig.1. Environment model used by [6].

We think that, the environment proposed in [6] suffers from two problems. The first problem has roots in an unrealistic assumption made in that paper. Fig. 1 and explanations given about, expresses that the whole existing agents in the environment have identical capabilities and only their position in the tree structure is different. Obviously this assumption doesn't seem accurate in the heterogeneous environment of grid in which the existing resources are heterogeneous and the capabilities are different. The second problem of [6] is that when a service is required for a user, primarily the resource supplier of this service is searched locally. If this service is found, the process of searching ends successfully; otherwise, the information in tables GACT and LACT should be searched to found the suitable resource. It seems that this method makes service discovery time very long and prolongs the time of achieving the load balancing.

Our proposed method considers above mentioned limitations to give a perfect solution. In order to solve the first problem, we consider different processing capabilities for the existing resources in the environment. This means that the agents that are located in different positions may have different processing power and even they possibly will have different functions. Since the existence of different capabilities in the agents leads to different services by them, consequently, the user can also receive appropriate responses from the environment and agents for his expected services. To address the second problem agents employ a specific data structure, called agent's identifier, to advertise their capabilities. An identifier consists of 16 bits which have been organized in 5 different fields, as shown in Fig. 2. The 1-bit **Busy/Idle** field indicates that the agent is either idle or busy just now. If the agent is busy the value of this bit will be 1 otherwise it will be 0. The 2-bit **Resource power** filed specifies the power of resources (agents). Larger numbers refers to more capabilities in the agents. By putting 3 in this field the agent is advertised as a powerful agent and 2 and 1 indicate average and weak resource powers respectively. It should be mentioned that the notions of powerful, average and weak may differ from one experiment to other one. The 4-bit **Node number** field is a

positive integer number that specifies the position of each resource (node number) in the tree hierarchy. **Unused** is a 1-bit field that is reserved for future possible extensions. And finally 8-bit **STAR** field (Successful Task Allocation Rate) indicates the number of tasks that have been served successfully in the current agents during past 5 seconds. The proposed environment can be observable in Fig. 3.

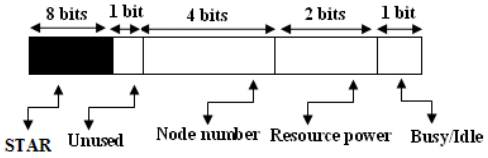

Fig.2. Format of identifier.

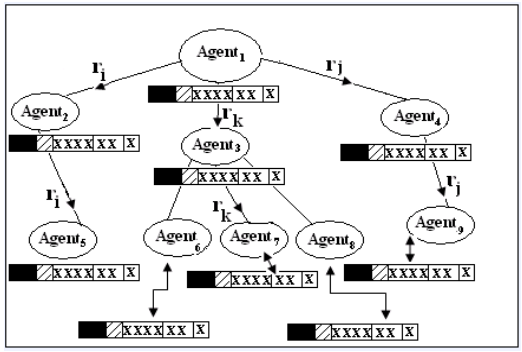

Fig.3. Proposed heterogeneous environment model.

The way of using this identifier and resource power has been explained by Algorithm 1. At the beginning, every existing agent in the environment fills various fields of its identifier by using local information. Then it sends the identifier to the head of its group. When the user sends its request to the environment in order to get service, this request primary is given to the head of the group or broker. Since the head of the group acts as a supervisor, it employs its sub-group to provide the requested services for the user. In this way broker first checks the Busy/Idle field of the agents exist in its subgroup. After finding idle resources, the header checks other fields of their identifiers to determine whether the agents have enough resources power to answer the request or not. Those agents which lack this specification are put aside and among the selected agents, those agents that have higher power to represent services and also those ones whose position are near to the head of the group have higher priorities. Finally user request is allocated to the resource with highest priority. It should be mentioned that all above steps for head of the group are exerted to all coordinators (C) which they also themselves conduct a sub-hierarchy. This is because all the requests from the head of the groups conduct toward the leaf nodes so that the requests can be answered by local resources. In this way the problem of bottleneck is not occurred. If two agents have same amount of processing capacity, the agent is selected that its node number is lower or equivalently is closer to the head of the group.

Algorithm 1. GLBAI Algorithm

```
1.  // Initialization Phase
2.  for i=1 to n do   // n is the number of agents
3.    {
4.        Create the identifier of agent_i  // Format of the 8-bit identifier is as in Fig. 2
5.        Fill various field of the identifier
6.        Send the identifier to the head of the group
7.     }
8.    ------------------------------------------------------------------------
9.  // User Request Submission
10. // if a user sends a request pass the request to the header (B according to Fig. 1)
11. // Header should check the sub-hierarchy  i.e C according to Fig. 1
12. for i=2 to n do
13.   {
14.       if agent_i[0] = 1 then  // agent_i is busy
15.         {
16.           ignore this agent
17.         }
18.       if agent_i[0] = 0 then  // agent_i is idle
19.         {
20.            check agent_i[1,2];
21.            if agent_i[1,2]=01 then
22.                  agent = powerful;
23.            else if agent_i[1,2]=10 then
24.                  agent= average;
25.            else
26.                  agent= weak;
27.            Select the resource that has more power
28.            if agent_i[1,2] = agent_j[1,2] // agents i and j have same priorities
29.              {
30.                check agent_i[3,4,5,6] and agent_j[3,4,5,6], find the location of each agent and
31.                give priority to agents that is closer
32.              }
33.           Check the STAR field and give more priority to agents with more successful  history
34.         }
35.      }
36.       Allocate the request to the agent that has the highest priority.
```

Simulation results and analysis

In this section, we demonstrate the simulation results of our proposed algorithm. The simulation has been done in the Gridsim environment [7]. As a comparison reference point for the proposed algorithm, we implemented the Cao algorithm in Gridsim environment and validated it through extensive simulation scenarios.

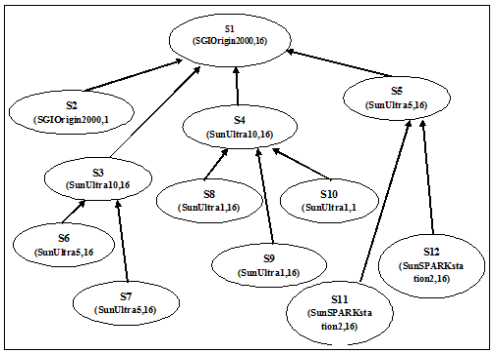

Fig.4. Grid environment used in our simulation.

A. Simulation setup

In contrast to Cao algorithm which assumes that all nodes (agents) in the environment have identical capabilities, we consider a more realistic grid environment in which various nodes (agents) have different capabilities i.e. they have different number of processors. As an example scenario consider the tree of Fig 4. In which there are 12 agents in 3 levels. Based on processing capacity each agent is labeled as weak, average or powerful agent, where their processing capability is assumed to be 200, 600 and 900 MIPS respectively. Table 1 shows the assigned processing capacity for each agent. Consider 50 tasks each one including 300 instructions enter to be served in the grid environment. The overhead involved in handling of each task is assumed to be 3 seconds that arises from communication delays. To show promise of the proposed

algorithm we simulate the environment once under GLBAI algorithm and then under Cao algorithm [6] to compare their performance by using Gridsim simulator.

B. Simulation results

By using the results generated by Gridsim it can be extracted that which task has been executed in each agent and how long it has taken. Using these results and according to equation (1) utilization of various resources are have been computed as shown in Table 2 for both GLBAI and Cao methods. According to this table average utilization of resources is 0.94 under GLBAI algorithm while Cao algorithm leads to average utilization of 0.80. This table shows that S_2, S_7 and S_8 have higher utilization comparing to other resources. This is reasonable since according to Table 1 these resources are the most powerful resources in the grid and hence their utilization are more effective than the other resources. On the other hand S_9 and S_{11} which are weak resources devote the minimum utilization. It should be mentioned that since S_1, S_3, S_4 and S_5 are head agents they don't attend in tasks execution and involves only in efficient allocation of arriving tasks to appropriate agents.

Table 1. Assumed Recourses power.

Resource number	Resource power
S_2, S_7, S_8	Powerful
S_6, S_{10}, S_{12}	Average
S_9, S_{11}	Weak

Table 2. Simulation results: resource utilization.

Agents Utilizations	Using GLBAI	Using Cao [6]
U_1	head of the groups	0.81
U_2	1	0.81
U_3	head of the groups	0.77
U_4	head of the groups	0.82
U_5	head of the groups	0.82
U_6	0.92	0.78
U_7	1	0.84
U_8	1	0.82
U_9	0.91	0.80
U_{10}	0.92	0.81
U_{11}	0.91	0.75
U_{12}	0.92	0.78

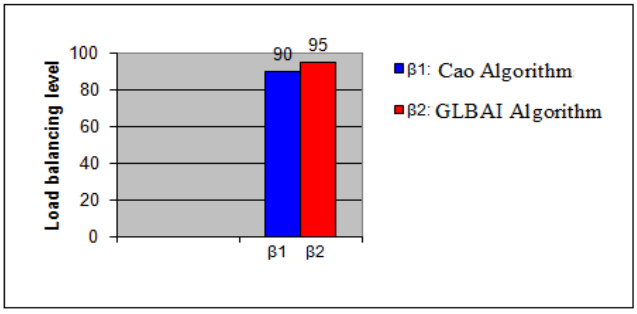

Fig.5. Load balancing levels of GLBAI and Cao Algorithms.

To address the load balancing issue in GLBAI and Cao approaches we apply equation (2)-(3) to Table 2 and extract the load balancing level i.e β. It is clear that when utilizations of various resources are near to each other load balancing level i.e. β will approach to 1. Figure 5 shows load balancing levels computed for GLBAI and Cao algorithms. We see that load balancing level GLBAI is 95 percent but in the Cao methods it is 90 percent. And finally simulation results show that the execution time of these submitted 50 tasks is 16.55 seconds for Cao method while GLBAI decreases it to 7.84 seconds.

(1) $$Ui = \frac{busy\ time\ for\ each\ agent}{total\ execution\ time} \qquad i = 1,2,....n$$

where: U – utilization of various resources.

(2) $$Uavg = \frac{\sum_{i=1}^{n} Ui}{n}$$

where: n – number of available resources, $Uavg$– average utilization.

(3) $$\beta = 1 - \frac{\sqrt{A}}{Uavg}$$

where: β– load balancing level $A = \sqrt{\frac{\sum_{i=1}^{n} (Uavg - Ui)^2}{n}}$

Conclusion and future works

This paper proposed a new method to reduce resource advertisement and discovery time and also improvement of load balancing level. It employed a special data structure called agent identifier for this purpose. The proposed method applied to a heterogeneous environment in which each node had a different processing capacity. The result of simulation showed that the exertion of proposed method can reduce the task execution time. Furthermore, it was shown that by this method we can improve the load balancing level and resources utilization.

REFERENCES

[1] Fei Y., Changjun J., Rong D., Jianjun Y.: "Grid resource management policies for load-balancing and energy-saving by vacation queuing theory", Elsevier Computers and Electrical Engineering, 2009.

[2] Shivaratri NG., Krueger P., Singhal M.: "Load distributing for locally distributed systems Computer", 1992.

[3] Zaki MJ., Parthasarathy WLS.: "Customized dynamic load balancing for a network of workstations", Journal of Parallel Distributed Computer, 1997.

[4] Li Y., Yang Y., Ma M., Zhou L.: "A hybrid load balancing strategy of sequential tasks for Grid computing environments", Elsevier, future generation computer systems, 2009.

[5] Yan K.Q., Wang S.C., Chang C.P., Lin b.: "A hybrid load balancing policy underlying grid computing environment", Elsevier Computer Standards & Interfaces, 2007.

[6] Cao J., Spooner D.P., Jarvis S.A., Nudd G.R.: "Grid load balancing using intelligent agents", Elsevier, Future Generation Computer Systems, 2005.

[7] Luther A., Nadiminti K., Buyya R.: "A. Net-based Enterprise Grid system and Framework-User Guide for Alchemi 1.0", Technical report, The university of Melborn, Australia, 2005.

[8] Cao J., Kerbyson D. J., Nudd G. R.: "Dynamic application integration using agent-based operational administration, International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, 2000.

[9] Cao J., Kerbyson D. J., Nudd G. R.: "High and mobile-agent systems, International Journal of Software Engineering and Knowledge Engineering", 2001.

[10] Mezmaz M., Melab N., Talbi E.G.: "An efficient load balancing strategy for Grid-based branch and bound algorithm", Elsevier parallel computing, 2007.

[11] Salehi M.A., Deldari H., Dorri B.M.:"Balancing Load in a Computational Grid Applying Adaptive Intelligent Colonies of Ants", Journal of informatica, 2008.

[12] Zikos S., Karatza H. D.: "Communication cost effective scheduling policies of non-clairvoyant jobs with load balancing in a grid", Elsevier Journal of Systems and Software, 2009.

***Authors**: prof. dr* Shahram Jamali, Department of Computer Engineering, University of Mohaghegh Ardabili, Ardabil, Iran, *E-mail:* jamali@iust.ac.ir; *Shiva RazzaghZadeh, Young Researchers Club, Ardabil Branch, Islamic Azad University, Ardabil, Iran, E-mail:* Shiva.razzaghzadeh@gmail.com,