ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(21), DOI: 10.17485/ijst/2016/v9i21/90840, June 2016* ISSN (Online) : 0974-5645

A Dynamic Load Balancing Algorithm for Computational Grid using Ant Colony Optimization

Hayyan Rajab* **and Kasem Kabalan**

Department of Computer Systems and Networks, Faculty of Informatics Engineering, Tishreen University, Lattakia, Syria;
hayyan.rajab@gmail.com, kasem.kabalan57@gmail.com

Abstract

Objective: To design and implement an algorithm for load balancing with convenient utilization of heterogeneous grid resources. **Methods:** In this paper, we introduce Ant based Dynamic Load Balancing Algorithm (ADLBA), a decentralized dynamic load balancing algorithm using Ant Colony Optimization (ACO), which selects the best resources to be allocated to the tasksconsidering economic cost, resources' capacity, and local load. **Results:** We used the Gridsim toolkit to evaluate the efficiency of ADLBA against the Randomized Algorithm (RA) with various number of tasks and resource allocation polices. Our study results show that ADLBA outperforms RA in terms of execution cost and total application execution time (makespan), and they also show that using time-shared allocation policy in the resources leads to better results in both algorithms. **Conclusion:** We found that ADLBA is suitable for grid users which aim to execute their applications quickly with lower cost.

Keywords: Ant Colony Optimization (ACO), Grid Computing, Gridsim, Load Balancing, Makespan

1. Introduction

Due to the availability of powerful and low cost computers over the internet, computing environments have been mapped from distributed to grid[1]. Grid is a distributed systemwhich gives the ability to share and coordinate the use of multi-owner and geographically distributed resources transparently regardless of their physical characteristics or locations by combing them into a single powerful system[2,3]. By using this technology, these resources can be utilized to serve large-scale applications which require a lot of computing power and data, e.g., meteorological simulations, research of DNA sequences, and data intensive applications[4].

There are two classes of grid systems: computational and data grids. In computational grids, compute cycles (i.e., processors) are the main resource; while the main resource of data grids is data which are distributed over various geographical locations[5].

In a grid system, the grid user submits the application to a resource broker which makes the grid details and complexities hidden from the users. The broker queriesa database of information about all grid resources called the Grid Information System (GIS). According to the result of this query, the broker schedules the application tasks by mapping them to the available resources, starts and manages their execution, and finally collects the results and resends them to the user[6,7].

1.1 Load Balancing Algorithms

In order to improve the performance and efficiency of a grid system, and to fulfill the requirements of its users, an effective strategy for load balancing while allocating resources to tasks is needed. Load balancing algorithms should focus on preventing situations whereunfair load distribution among resources takes place[8]. Objective functions of the aforementioned algorithms areeither application-centric or resource-centric[9]. Application

* *Author for correspondence*

centric objective functions aim to enhance the performance of a single application that concerns about time in terms of makespan (total application execution time) and economic cost. Resource-centric objective functions operate at the system level and focus on optimizing the performance of resources considering resource utilization and economic profit.

Load balancing algorithms are categorized into: Centralized and decentralized algorithems[10]. In centralized algorithms, all users submit their tasks to a single scheduler which is responsible for distributing them on the available resources. While in decentralized algorithms, each user has its own scheduler.

Furthermore, Load balancing algorithms are either static or dynamic[11]. In static algorithms, load balancing decisions are made when the estimation of resource requirements is done at compile time. Dynamic algorithms allocate/reallocate resources at runtime according to information about the current distribution of the load, therefore, some tasks can be migrated among resources. Due to the characteristics of grid computing, using dynamic algorithms can improve system performance over static algorithms, if the extra cost overhead of gathering and maintaining information is kept within reasonable limits[12].

The implementation of the following policies defines different Load balancing paradigms[13]: Information policy: Specifies when, what, and from where the collecting of workload information is performed. Triggering policy:chooses the suitable time to run a process for load balancing. Resource type policy:assorts resources into servers and receivers of tasks based on their current load. Location policy: Searches for a proper partner for a receiver/server using the results of resource type policy. Selection policy: Chooses the tasks that should be migrated from heavily-loaded resources to lightly-loaded resources. Some actions change the load configuration in Grid environment such as[14]: Arrival of a new task, finishing the execution of a task, and adding a new resource or withdrawal of any existing resource.

1.2 Ant Colony Optimization

The conduct of natural systems inspired the researchers to introduce a novel behavioral and computational model called Ant Colony Optimization (ACO) to solve combinatorial optimization problems[15]. Ants use a chemical substance called a pheromone to communicate with each other. They move randomly but if they find a pheromone trail on their way, there is a big chance that they follow it, lay down their pheromone, and reinforce this pathway. Ants' behavior helps to build the shortest path from the nest to food.

In this paper, we presentan ant based algorithm for load balancing in grid. This research work aims to improve the way ants (i.e., tasks) search for the best resources in terms of minimizing execution cost and makespan, while balancing the load among available resources.

The rest of paper is organized in the following manner: Section 2 gives a detailed literature review. Section 3 describes the proposed algorithm concept and design. Section 4 shows the simulation experiments and results, and Section 5 includes conclusion and future work.

2. Related Work

Although the load balancing problem of the traditional distributed systems has been studied, the dynamic nature and heterogeneity of grid resources make this problem more difficult to be solved. Many researchers proposed their algorithms to handle this issue.

In[14], a new load balancing mechanism was proposed. This algorithm depends on a new method for implementing the information policy. Instead of collecting information periodically, the proposed scheme collects information using activity based approach which adapts to the changes that take place in the system as soon as possible.

Many algorithms regarding ant colony optimization have been proposed. In[16], the researchers introduced a load balancing paradigm which works as follows: an ant is initialized once a task is submitted to a local node. This ant moves from one node to another collecting their load information, adding it to its history and updating the load information table of these nodes. Finally, the task is delivered to the node with the lowest load, after a set number of steps.

In other approaches, pheromones are associated with resources rather than paths. They represent the processing capability of the resources. Tasks are allocated to the resources with the highest pheromones. According totask status at a resource, the pheromone value decreases or increases[17–19].

In[20], an algorithm for load balancing was introduced. In this approach an ant in an overloaded resource starts

searching for the best resource (the one with the highest pheromone value) to deliver the overloaded task to it. The ant aims to locate the shortest path to reduce the cost of load balancing operations.

In this paper, we formulate a decentralized dynamic load balancing algorithm for computational grid environments. Compared to the previous related work, our proposed algorithm has the following advantages:

- Price factor and local load factor are considered.
- A new method for selecting the resource which will be allocated is proposed.
- Task migration among resources in case an activity takes place and changes the load configuration in grid environment, is supported.

3. Ant based Dynamic Load Balancing Algorithm (ADLBA):

In ADLBA, the pheromone is associated with the resource rather than the path. Pheromone value changes according to task status at resources. Instead of looking for the best path, ADLBA looks for the best resource to allocate the tasks to. The best resource is the one which has the highest processing capacity, lowest cost, and lowest local load. The purpose of this algorithm is to decrease execution cost and makespan. The notations used in the description of ADLBA are illustrated in Table 1.

Table 1. Notations used

K	Number of available resources
N	Number of processing elements in the resource
M	Execution speed of the resource (million instructions per second(MIPS))
R_c	Cost of using resource per second
$load_{local}$	Local load on the resource
PH_i^{init}	Initial pheromone associated with the resource
PH_i^t	Current pheromone associated with the resource
T	Set of the application tasks
AL	The application length (million instructions (MI))
C	Complexity of the assigned task (million instructions)
$PH_{average}^t$	The current average of pheromone Values
σ^t	The current standard deviation of pheromone Values

The work steps of ADLBA are shown below:

Step 1:[Initialize Pheromone Value of Each Resource]

For every new resource R_i registered to Grid Information Server, the initial pheromone PH_i^{init} is given as:

Formula (1): $PH_i^{init} = [N \times M \times (1 - load_{local})] / R_c$

Formula (2): $PH_i^{t=0} = \left[PH_i^{init} \times AL\right] / \sum_{j=0}^{k} PH_j^{init}$

Step 2: [Select Task]

Repeat 3 to 5 while (T ≠ θ)

Select task t from task set T.

Step 3: [Select Resource]

Determine the resource R_i for task t which has the highest pheromone value.

Step 4: [Schedule Task to Selected Resource]

Submit task t to R_i and removeitfrom T:

T = T - {t}

Step 5: [Update Pheromone]

For every resource R_i allocated for a task, the pheromone is decreased by C:

Formula (3): $PH_i^{new} = PH_i^{old} - C$

If (Task_Status= Failure) add it again to T:

T = T + {t}

Go to 2

If (Task_Status= Succesful_Complete at any resource R_i) then increase the pheromone of R_i by C:

Formula (4): $PH_i^{new} = PH_i^{old} + C$

Go to 6

Step 6: [Check Imbalance State and Migrate Tasks]

Calculate $PH_{average}^t$ and σ^t.

While ($PH_i^t > PH_{average}^t + \sigma^t$) migrate tasks from R_j to R_i where ($PH_j^t < PH_{average}^t - \sigma^t$), and update pheromone values as follows:

Formula (5): $PH_i^{new} = PH_i^{old} - C$

Formula (6): $PH_j^{new} = PH_j^{old} + C$

Step 7: EXIT.

In Formula (1), $load_{local}$ is a value in the range [0, 1] which represents the local load of the resource R_i and is calculated according to the number of tasks assigned by a local user, where (0) means that a resource is idle, and (1) identifies that it is busy. After the initial pheromone values are calculated, Formula (2) is used to transform them into million instructions unit, where the ratio of each resource pheromone value to the total application length equals the ratio of its initial value to the sum of all initial pheromone

values. This method ensures that the pheromone does not have a negative value during the execution time. When a task execution is completed by a resource, the pheromone value is increased by (C) which is the complexity of the task. Consequently, this trail that the ant (task) lays down increases the probability of choosing this resource from another ant.

As shown in Step (6), when an activity (completion of a task) occurs, the scheduler calculates the upper threshold ($PH^{t}_{average}+\sigma^{t}$) and the lower threshold ($PH^{t}_{average}-\sigma^{t}$) to determine the load status on resources, and migrates tasks from resources with pheromone values smaller than the lower threshold (heavy loaded) to resources with pheromone values bigger than the upper threshold (lightly loaded). This migration is performed under the condition that the new pheromone value of a receiver is not smaller than the new pheromone value of a sender. Although it handles only a single activity, this approach can be used and improved to handle other activities. The load balancing policies implemented in ADLBA are illustrated in Table 2.

Table 2. Load balancing policies in ADLBA

Information policy	Load Balancing information is collected using activity based approach
Triggering, resource type and location policies	Based on pheromone values
Selection policy	Task is selected for migration using task length as criteria

4. Simulation Setup and Results

We used Gridsim, a java-based discrete event simulation toolkit, for our experimental study. This toolkit enables researchers to modeland simulate heterogeneous resources and users. It supports creation of application tasks, distributing and managingthem on the available resources[21]. The simulation parameters for scheduling and resource characteristics are shown in Table 3.

Allocation policy of the resource can be either time-shared or space-shared. Time-shared policy is a round-robin allocation method in which a Processing Element (PE) can be shared by several tasks (gridlets) using time slots, therefore a task execution can start immediately whenever it arrives. Space-shared is a first-come-first-serve allocation policy where a task can't start its execution and has to wait in a queue if there is no a free PE.

We assume that each user submits an application which consists of a number of independent tasks with different lengths, and the order of execution tasks is not significant. In order to test the efficiency of ADLBA, we compared its execution cost and makespan with the Randomized Algorithm (RA).

Table 3. Simulation parameters

Simulation Parameters	Value
Number of users	1 or 5
Number of resources	50
Number of machines per resource	1
Number of PEs per machine	1-4
Processing capacity of each PE	10 or 50 MIPS
Allocation policy	Time-Shared or Space-Shared
Cost of using resource per second	0.1 to 0.5 G$
Local load	0.2 -0.8
Number of tasks(ants) per application	50-200
Task length	100-20000 MI
Input file size	100 + (10% to 40%)
Output file size	100 + (10% to 50%)

4.1 Performance Evaluation Criteria

4.1.1 Execution Cost

Execution cost for a single user is given by the following formula:

Formula (7): $Execution\text{Cost} = \sum_{i=1}^{N} C_{R_i} \times P_{R_i}$

Where N is number of resources, C_{R_i} is total computation time of assigned tasks at the resource R_i, and P_{R_i} is cost of using the resource per second.

4.1.2 Makespan

The makespan for a single user is measured from the moment the first task is submitted, until the last task returns.

4.2 Results

4.2.1 Scenario 1

Here, the number of users is set to (1), and resources use time-shared as an allocation policy. The performance of (ADLBA) is compared with (RA) for a various number of tasks with increased lengths. The number of the submitted tasks represents the system load. As shown in Figures (1,2), when the system load increases the execution

cost and makespan increase in the both algorithms. The results show that ADLBA performs better than RA, and the differnce in performance between the both algorithm grows as the number of tasks increases.

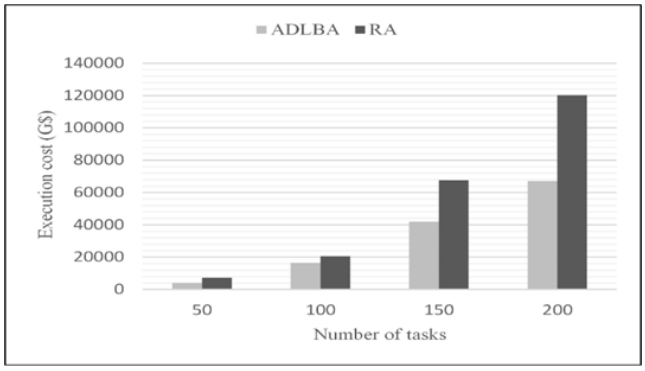

Figure 1. Number of tasks vs execution cost in time-shared allocation.

Figure 2. Number of tasks vs makespan in time-shared allocation.

The average improvement is: 40 % for execution cost, and 32% for makespan.

4.2.2 Scenario 2

This is a single user scenario where space-shared allocation policy is used. Figures (3,4) show that the higher the load the higher the execution cost and makespan of both algorithms. Results show that ADLBA out performs RA, and the differnce in perfrmance between both schemes increases as the number of tasks increases. When compared with the results of the previous scenario, we notice that using time-shared allocation policy in the resources leads to better results in both algorithms, this is because in space-shared allocation policy if all PEs are busy, the task has to wait more before its execution is initiated.

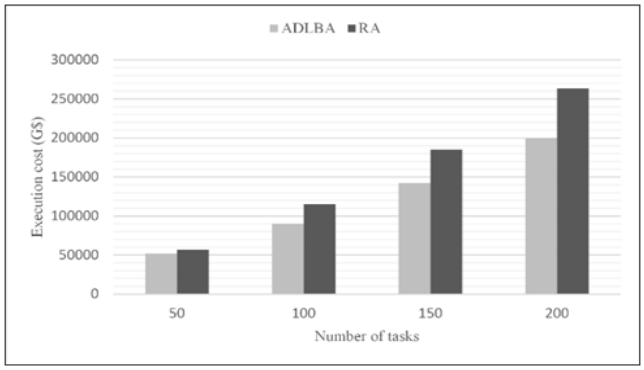

Figure 3. Number of tasks vs execution cost in space-shared allocation.

Figure 4. Number of tasks vs makespan in space-shared allocation.

The average improvement in execution Cost is 22 %, and in Makespan is 34%.

4.2.3 Scenario 3

In this scenario, the number of users is 5 which means that there are 5 schedulers (resource brokers) which not corporate with each other. We compare the performance of ADLBA against the randomized algorithm using Time-shared or Space-Shared as an allocation policy in the resources. As shown in Figures 5, 6. The results show that ADLBA performs better than RA, and using Time-Shared leads to better results.

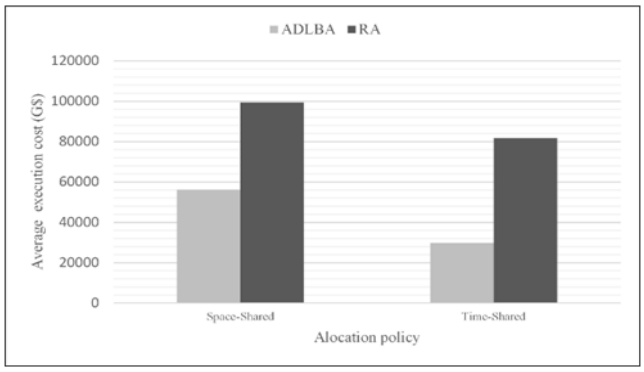

Figure 5. Allocation policy vs average execution cost.

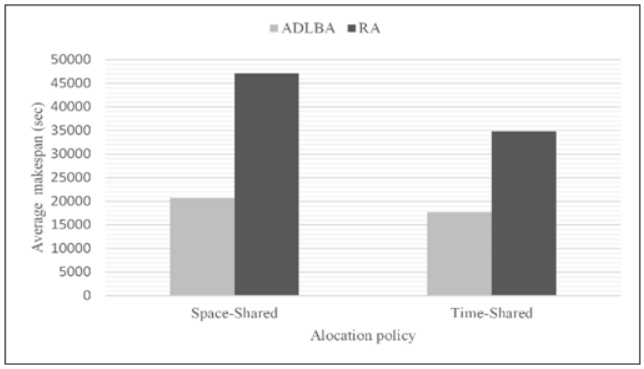

Figure 6. Allocation policy vs average makespan.

The average improvement in Execution Cost is 63 %, and in Makespan is 49%.

5. Conclusion

In this paper, ADLBA, a decentralized and dynamic load balancing algorithm in the grid environment is proposed. This work aims at minimizing makespan and execution cost, and takes into account the heterogeneity, cost, and local load of computational resources. We studied tasks that are independent of each other. In the future, we will improve ADLBA so it can handle several issues such as network delay, Quality of Service (QoS) requirements, and fault tolerance.

In this research, we considered the local load factor ($load_{local}$) only at scheduling time, we plan to make our algorithm adapt to its changes during execution.

6. References

1. Kaur P, Singh H. Dynamic Load balancing in grid environment using adaptive computing. International Conference on Recent Trends of Computer Technology in Academia (ICRTCTA-2012); 2012. p. 625–32.
2. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications. 2001 Aug;15(3):200–2.
3. Devi KN, Tamilarasi A. Dynamic scheduling in grid environent with the improvement of fault tolerant level. Indian Journal of Science and Technology. 2015 Apr; 8(8):507–15.
4. Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications. 2000 Apr; 3(3):187–200.
5. Salleh S, Zomaya AY. Scheduling in parallel computing systems: Fuzzy and annealing techniques. New York: Springer Science and Business Media; 2012.
6. Sharma R, Soni VK, Mishra MK, Bhuyan P. A survey of job scheduling and resource management in grid computing. International Journal of Computer, Electrical, Automation, Control and Information Engineering. 2010; 4(4):736–41.
7. Seema K, Ku G, Kumar RM, Rahul S. Improved hybrid scheduling algorithm for grid infrastructure. Indian Journal of Science and Technology. 2015 Dec; 8(35):1–6.
8. Xu C, Lau FC. Load balancing in parallel computers: Theory and practice. New York: Springer Science and Business Media; 1997.
9. Zhu Y, Ni LM. A survey on grid scheduling systems. Hong Kong: University of science and Technology; 2003.
10. Kandagatla C. Survey and taxonomy of grid resource management systems. Austin: University of Texas; 2003. p. 1–12.
11. Zaki MJ, Li W, Parthasarathy S. Customized dynamic load balancing for a network of workstations. 1996 Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing; Syracuse, NY, USA. 1996. p. 282–91.
12. Von Laszewski G, Foster I, Gawor J, Lane P, Rehn N, Russell M. Designing grid-based problem solving environments and portals. 2001 Proceedings of the 34th Annual Hawaii International Conference on System Sciences; Maui, HI, USA. 2001.
13. Nath RK. Efficient load balancing algorithm in grid environment. Patiala: Thapar University; 2007.
14. Srivastava PK, Gupta S, Yadav DS. Improving performance in load balancing problem on the grid computing system. International Journal of Computer Applications. 2011 Feb; 16(1):6–10.
15. Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoretical Computer Science. 2005 May; 344(2-3):243–78.
16. Ludwig SA, Moallem A. Swarm intelligence approaches for grid load balancing. Journal of Grid Computing. 2011 Sep; 9(3):279–301.
17. Goyal SK, Singh M. Adaptive and dynamic load balancing in grid using ant colony optimization. International Journal of Engineering and Technology. 2012; 4(4):167–74.

18. Rohil H, Kalyan S. A heuristic based load balancing algorithm. IJCEM. 2012 Nov; 15(6):56–61.
19. Sharma D, Sharma K, Dalal S. Optimized load balancing in grid computing using tentative ant colony algorithm. International Journal of Recent Research Aspects. 2014 Jun; 1(1):35–9.
20. Nadimi-Shahraki MH, Fard ES, Safi F. Efficient load balancing using ant colony optimization. Journal of Theoretical and Applied Information Technology. 2015 Jul; 77(2):253–8.
21. Buyya R, Murshed M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: Practice and experience. 2002 Nov; 14(13-15):1175–20.