Balancing Load in a Computational Grid Applying Adaptive, Intelligent Colonies of Ants

Mohsen Amini Salehi
Department of Software Engineering, Faculty of Engineering, Islamic Azad University, Mashhad Branch, Iran
E-mail: Amini@mshdiau.ac.ir

Hossein Deldari
Department of Software Engineering, Faculty of Engineering Ferdowsi University of Mashhad, Iran
E-mail: hd@um.ac.ir

Bahare Mokarram Dorri
Management and Planning Organisation of Khorasan, Mashhad, Iran
E-mail: mokarram@mpo-kh.ir

Keywords: Grid computing, Load balancing, Ant colony, Agent-based Resource Management System (ARMS)

Received: July 1, 2007

Load balancing is substantial when developing parallel and distributed computing applications. The emergence of computational grids extends the necessity of this problem. Ant colony is a meta-heuristic method that can be instrumental for grid load balancing. This paper presents an echo system of adaptive fuzzy ants. The ants in this environment can create new ones and may also commit suicide depending on existing conditions. A new concept called Ant level load balancing is presented here for improving the performance of the mechanism. A performance evaluation model is also derived. Then theoretical analyses, which are supported with experiment results, prove that this new mechanism surpasses its predecessor.

Povzetek: Metoda inteligentnih mravelj je uporabljena na problemu razporejanju bremen.

1 Introduction

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

The resource manager is an important infrastructural component of a grid computing environment. Its overall aim is to efficiently schedule applications needing utilization of available resources in the grid environment. A grid resource manager provides a mechanism for grid applications to discover and utilize resources in the grid environment. Resource discovery and advertisement offer complementary functions. The discovery is initiated by a grid application to find suitable resources within the grid. Advertisement is initiated by a resource in search of a suitable application that can utilize it. A matchmaker is a grid middleware component which tries to match applications and resources. A matchmaker may be implemented in centralized or distributed ways. As the grid is inherently dynamic, and has no boundary [1], so the distributed approaches usually show better results [2] and are also more scalable. A good matchmaker (broker) should uniformly distribute the requests, along the grid resources, with the aid of load balancing methods.

As mentioned in [1], the grid is a highly dynamic environment for which there is no unique administration. Therefore, the grid middleware should compensate for the lack of unique administration.

ARMS is an agent-based resource manager infrastructure for the grid [3, 4]. In ARMS, each agent can act simultaneously as a resource questioner, resource provider, and the matchmaker. Details of the design and implementation of ARMS can be found in [2]. In this work, we use ARMS as the experimental platform.

Cosy is a job scheduler which supports job scheduling as well as advanced reservations [5]. It is integrated into ARMS agents to perform global grid management [5]; Cosy needs a load balancer to better utilize available resources. This load balancer is introduced in part 3.

The rest of the paper is organized as follows: Section 2 introduces the load balancing approaches for grid resource management. In Section 3, ant colony optimization and self-organizing mechanisms for load balancing are discussed. Section 4 describes the proposed mechanism. Performance metrics and simulation results are included in Section 5. Finally, the conclusion of the article is presented as well as future work related to this research.

2 Load balancing

Load balancing algorithms are essentially designed to spread the resources' load equally thus maximizing their utilization while minimizing the total task execution time [7]. This is crucial in a computational grid where the most

pressing issue is to fairly assign jobs to resources. Thus, the difference between the heaviest and the lightest resource load is minimized.
A flexible load sharing algorithm is required to be general, adaptable, stable, scalable, fault tolerant, transparent to the application and to also induce minimum overhead to the system [8]. The properties listed above are interdependent. For example, a lengthy delay in processing and communication can affect the algorithm overhead significantly, result in instability and indicate that the algorithm is not scalable.
The load balancing process can be defined in three rules: the location, distribution and selection rule [7]. The location rule determines which resource domain will be included in the balancing operation. The domain may be local, i.e. inside the node, or global, i.e. between different nodes. The distribution rule establishes the redistribution of the workload among available resources in the domain, while the selection rule decides whether the load balancing operation can be performed preemptively or not [7].

2.1 Classification of load balancing mechanisms

In general, load balancing mechanisms can be broadly categorized as centralized or decentralized, dynamic or static [10], and periodic or non-periodic [11].
In a centralized algorithm, there is a central scheduler which gathers all load information from the nodes and makes appropriate decisions. However, this approach is not scalable for a vast environment like the grid. In decentralized models, there is usually not a specific node known as a server or collector. Instead, all nodes have information about some or all other nodes. This leads to a huge overhead in communication. Furthermore, this information is not very reliable because of the drastic load variation in the grid and the need to update frequently.
Static algorithms are not affected by the system state, as their behaviour is predetermined. On the other hand, dynamic algorithms make decisions according to the system state. The state refers to certain types of information, such as the number of jobs waiting in the ready queue, the current job arrival rate, etc [12]. Dynamic algorithms tend to have better performance than static ones [13].
Some dynamic load balancing algorithms are adaptive; in other words, dynamic policies are modifiable as the system state changes. Via this approach, methods adjust their activities based on system feedback [13].

3 Related works

Swarm intelligence [14] is inspired by the behaviour of insects, such as wasps, ants or honey bees. The ants, for example, have little intelligence for their hostile and dynamic environment [15]. However, they perform incredible activities such as organizing their dead in cemeteries and foraging for food. Actually, there is an indirect communication among ants which is achieved through their chemical substance deposits [16].
This ability of ants is applied in solving some heuristic problems, like optimal routing in a telecommunication network [15], coordinating robots, sorting [17], and especially load balancing [6, 9, 18, 19].
Messor [20] is the main contribution to the load balancing context.

3.1 Messor

Messor is a grid computing system that is implemented on top of the Anthill framework [18].
Ants in this system can be in Search–Max or Search–Min states. In the Search–Max state, an ant wanders around randomly until it finds an overloaded node. The ant then switches to the Search–Min state to find an underloaded node. After these states, the ant balances the two overloaded and underloaded nodes that it found. Once an ant encounters a node, it retains information about the nodes visited. Other ants which visit this node can apply this information to perform more efficiently. However, with respect to the dynamism of the grid, this information cannot be reliable for a long time and may even cause erroneous decision-making by other ants.

3.2 Self-Organizing agents for grid load balancing

In [6], J.Cao et al propose a self-organizing load balancing mechanism using ants in ARMS. As this mechanism is simple and inefficient, we call it the “seminal approach”. The main purpose of this study is the optimization of this seminal mechanism. Thus, we propose a modified mechanism based on a swarm of intelligent ants that uniformly balance the load throughout the grid.
In this mechanism an ant always wanders ‘2m+ 1’ steps to finally balance two overloaded and underloaded nodes.
As stated in [6], the efficiency of the mechanism highly depends on the number of cooperating ants (n) as well as their step count (m). If a loop includes a few steps, then the ant will initiate the load balancing process frequently, while if the ant starts with a larger m, then the frequency of performing load balancing decreases. This implies that the ant’s step count should be determined according to the system load. However, with this method, the number of ants and the number of their steps are defined by the user and do not change during the load balancing process. In fact, defining the number of ants and their wandering steps by the user is impractical in an environment such as the grid, where users have no background knowledge and the ultimate goal is to introduce a transparent, powerful computing service to end users.
Considering the above faults, we propose a new mechanism that can be adaptive to environmental conditions and turn out better results. In the next section, the proposed method is described.

4 Proposed method

In the new mechanism, we propose an echo system of intelligent ants which react proportionally to their conditions. Interactions between these intelligent,

autonomous ants result in load balancing throughout the grid.
In this case, an echo system creates ants on demand to achieve load balancing during their adaptive lives. They may bear offspring when they sense that the system is drastically unbalanced and commit suicide when they detect equilibrium in the environment. These ants care for every node visited during their steps and record node specifications for future decision making. Moreover, every ant in the new mechanism hops 'm' steps (the value of 'm' is determined adaptively) instead of '2m+1'. At the end of the 'm' steps, 'k' overloaded are equalized with 'k' underloaded nodes, in contrast to one overloaded with one underloaded according to the previous method. This results in an earlier convergence with fewer ants and less communication overhead.
In the next sections, the proposed method is described in more detail.

4.1 Creating ants

If a node understands that it is overloaded, it can create a new ant taking only a few steps to balance the load as quickly as possible. Actually, as referred in [2], neighbouring agents, in ARMS, exchange their load status periodically. If a node's load is more than the average of its neighbours, for several periods of time, and it has not been visited by any ant during this time, then the node creates a new ant itself to balance its load throughout a wider area.
Load can be estimated several ways by an agent to distinguish whether a node is overloaded or not. For the sake of comparison with similar methods, the number of waiting jobs in a node is considered the criterion for load measurement.

4.2 Decision-making

Every ant is allocated to a memory space which records specifications of the environment while it wanders. The memory space is divided into an underloaded list (Min List) and an overloaded list (Max List). In the former, the ant saves specifications of the underloaded nodes visited. In the latter, specifications of the overloaded nodes visited are saved.
At every step, the ant randomly selects one of the node's neighbours.

4.2.1 Deciding algorithm

After entering a node, the ant first checks its memory to determine whether this node was already visited by the ant itself or not. If not, the ant can verify the condition of the node, i.e. overloaded, underloaded or at an equilibrium, using its acquired knowledge from the environment.
As the load quantity of a node is a linguistic variable and the state of the node is determined relative to system conditions, decision making is performed adaptively by applying fuzzy logic [21, 22].
To make a decision, the ant deploys the node's current workload and the remaining steps as two inputs into the fuzzy inference system. Then, the ant determines the state of the node, i.e. Max, Avg or Min.

The total average of the load visited is kept as the ant's internal knowledge about the environment. The ant uses this for building membership functions of the node's workload, as shown in Figure1.a. The membership functions of Remain steps and Decide, as the output, are on the basis of a threshold and are presented in Figures 1.b, 1.c:

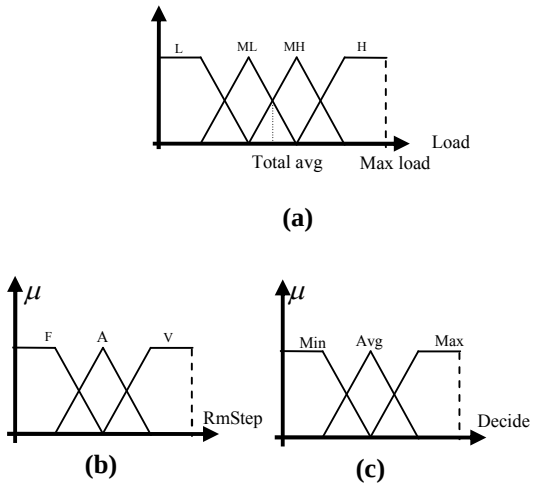

Figure 1: Membership functions of fuzzy sets
a) The Node's Load, b) Remain steps, c) Decide.

The inference system can be expressed as the following relation:

$$R_A : Load < L, ML, MH, H > * RmStep < F, A, V > \rightarrow Decide < Min, Avg, Max > \quad (1)$$

Where L, ML, MH, H in Figure1.a indicates Low, Medium Low, Medium High, High respectively and F, A, V in Figure 1.b indicates Few, Average and Very.
Thus, the ant can make a proper decision. If the result is "Max" or "Min", the node's specifications must be added to the ant's max-list or the min-list. Subsequently, the corresponding counter for Max, Min, or Avg increases by one. These counters also depict the ant's knowledge about the environment. How this knowledge is employed is explained in the next sections.

4.2.2 Ant level load balancing

In the subtle behaviour of ants and their interactions, we can see that when two ants face each other, they stop for a moment and touch tentacles, probably for recognizing their team members. This is what inspired the first use of ant level load balancing.
With respect to the system structure, it is probable that two or more ants meet each other on the same node. As mentioned earlier, each of these ants may gather specifications of some overloaded and underloaded nodes. The amount of information is not necessarily the same for each ant, for example one ant has specifications of four overloaded and two underloaded while the other has two overloaded and six underloaded nodes in the same

position. In this situation, ants extend their knowledge by exchanging them. We call this *"ant level load balancing."* In the aforementioned example, after ant level load balancing of the two co-positions, the ants have specifications of three overloaded and four underloaded nodes in their memories. This leads to better performance in the last step, when the ants want to balance the load of 'k' overloaded with 'k' underloaded nodes. This operation can be applied to more than two ants.

Actually, when two or more co-positioned ants exchange their knowledge, they extend their movement radius to a bigger domain, thus improving awareness of the environment. Another idea is taken from the ant's pheromone deposits while travelling, which is used by ants to pursue other ants. This is applied in most ant colony optimization problems [23, 24]. There is, however, a subtle difference between these two applications. In the former the information retained by the ant may become invalid over time. This problem can be solved by evaporation [23, 24]. Evaporation, however, is not applicable in some cases, e.g. in the grid, where load information varies frequently. On the other hand, in the latter application, the knowledge exchanged is completely reliable.

4.2.3 How new ants are created

In special conditions, particularly when the its life span is long, the ant's memory may fill up, even though the ant may still be encountering nodes which are overloaded or underloaded. In this situation, if a node is overloaded, the ant bears a new ant with predefined steps. If encountering an underloaded node, the ant immediately exchanges the node's specification with the biggest load on the list of underloaded elements. This results in better balancing performance and adaptability to the environment. Here, adaptability translates into increasing the number of ants automatically, whenever there is an abundance of overloaded nodes.

4.3 Load balancing, starting new itineration

When its journey ends, the ant has to start a balancing operation between the overloaded (Max) and underloaded (Min) elements gathered during its roaming. In this stage, the number of elements in the Max-list and Min-list is close together (because of ant level load balancing). To achieve load balancing, the ant recommends underloaded nodes to the overloaded nodes and vice versa. In this way, the amount of load is dispersed equally among underloaded and overloaded nodes.

After load balancing, the ant should reinitiate itself to begin a new itineration. One of the fields that must be reinitiated is the ant's step counts. However, as stated in previous sections, the ant's step counts (m) must be commensurate to system conditions [6]. Therefore, if most of the visited nodes 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. This knowledge should be based on the number of overloaded, underloaded and equilibrium nodes visited during the last itineration.

Because of fuzzy logic power in the adaptation among several parameters in a problem [22] and the consideration of the step counts (m) as a linguistic variable, e.g. short, medium, long, it is rational to use fuzzy logic for determining the next itineration step counts.

Actually, this is an adaptive fuzzy controller which determines the next itineration step counts (NextM, for short) based on the number of overloaded, underloaded and equilibrium nodes visited, along with the step counts during the last itineration (LastM). In other words, the number of overloaded, underloaded and equilibrium nodes encountered during the LastM indicate the recent condition of the environment, while the LastM, itself, reports the lifetime history of the ant.

The membership functions of the fuzzy sets are shown in Figure 2.

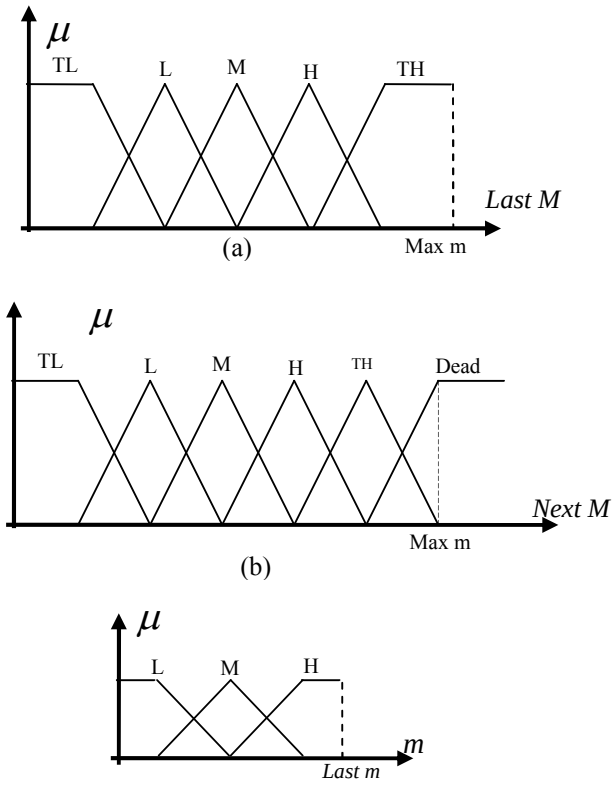

Figure 2: Membership functions of fuzzy sets
a) Last m, b) Next m, c) count of Max, Min and Average nodes visited

Where TL, L, M, H, TH shows Too Low, Low, Medium, High and Too High in Figure 2.a, 2.b and L, M, H indicates Low, Medium and High in Figure 2.c.

This fuzzy system can be displayed as a relation and a corresponding function as follows:

$$R_B : MaxCount<l,m,h> * MinCount<l,mh> * Avg<l,m,h> \rightarrow NextM<TL,L,M,H,TH,Dead> \quad (2)$$

$$f(x) = \frac{\sum_{r=1}^{135} \bar{y}^r * \prod_{i=1}^{4} \mu_{B_i^r}(x_i)}{\sum_{r=1}^{135} \prod_{i=1}^{4} \mu_{B_i^r}(x_i)} \quad (3)$$

Where x_i shows the input data into the system, $\bar{y}^r$ is the centre of the specific membership function declared in rule r. $\mu_{B_i^r}(x_i)$ indicates the membership value of the i th input in membership functions of the r th rule. In this inference system, we also have 4 inputs and 135 rules defined, as stated in (3).

In this system, a large number of underloaded and, especially, equilibrium elements indicate equilibrium states. Consequently, the NextM should be prolonged, thus lowering the load balancing frequency. One can say that, if an ant's step counts extend to extreme values, its effect tends to be zero. Based on this premise, we can conclude that an ant with overly long step counts does not have any influence on the system balance. Rather, the ant imposes its communication overhead on the system. In this situation, the ant must commit suicide. This is the last ring of the echo system. Therefore, if the *NextM* is fired in the *"Dead"* membership function, the ant does not start any new itineration.

Below is a diagram exhibiting the ant's behaviour in different environmental conditions. Figure 3.a shows the relation between the *LastM* and the amount of overloaded nodes visited, while Figure 3.b illustrates the relation between the *Last*M and the number of equilibrium nodes visited.

5 Performance valuations

In this section, several common statistics are investigated, which show the performance of the mechanism.

5.1 Efficiency

To prove that the new mechanism increases efficiency, it should be compared with the mechanism described in [4]. First, we introduce some of the most important criteria in load balancing:

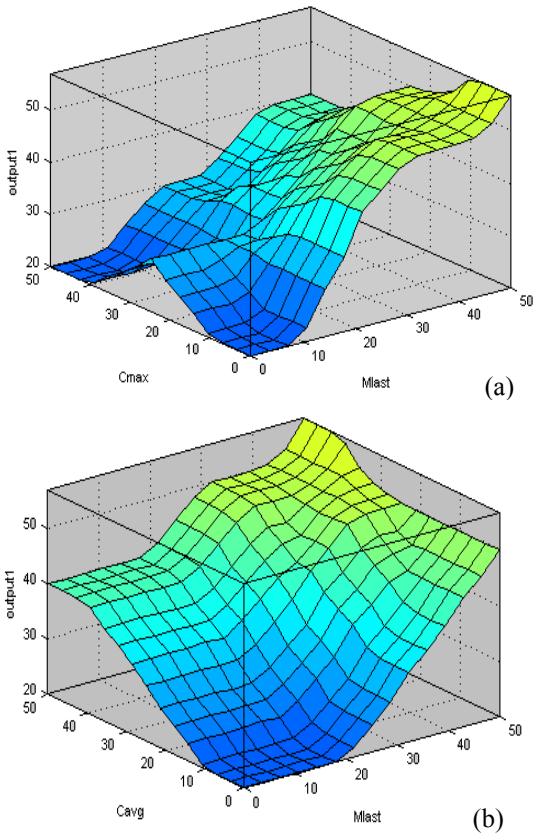

Figure 3: Schematic view of adaptive determining of next itineration step counts. a) LastM –MaxCount–output, b) LastM – avgCount–output

Let P be the number of agents and Wpk where (p: 1, 2... P) is the workload of the agent p at step k. The average workload is:

$$\overline{W_k} = \frac{\sum_{p=1}^{P} W_{pk}}{P} \quad (4)$$

The mean square deviation of Wpk, describing the load balancing level of the system, is defined as:

$$L_k = \sqrt{\frac{\sum_{p=1}^{P} (\overline{W_k} - W_{pk})^2}{P}} \quad (5)$$

Finally, The system load balancing efficiency (e) is defined as:

$$e_k = \frac{L_0 - L_k}{C_k} \quad (6)$$

Where ek means efficiency at step k and Ck is the total number of agent connections that have achieved a load balancing level Lk. To compare the efficiency of these two mechanisms, we should consider $e_{k_{new}} / e_{k_{Trad}}$.

As L0 indicates the load balancing level at the beginning of the load balancing process and is also equal in both new

and seminal mechanisms, we shall discuss the value of Lk. For the sake of simplicity, assume that every node gets to $\overline{W_k}$ after the balancing process and no longer requires balancing, i.e.

$$\overline{W_k} - W_{pk} = 0 \quad (7)$$

On the other hand, after the k stage, if the memory space considered for overloaded and underloaded elements is equal to 'a' (a>2), then we have ka elements balanced:

$$L_{k_{new}} = \sqrt{\frac{\sum_{p=1}^{p-ka}(\overline{W_k} - W_{pk})^2}{P}} \quad (8)$$

While in the seminal approach we have:

$$L_{k_{Trad}} = \sqrt{\frac{\sum_{p=1}^{p-2k}(\overline{W_k} - W_{pk})^2}{P}} \quad (9)$$

If we suppose that a>2, we can conclude:

$$P - 2k > P - ka \quad (10)$$

After the k stages, the difference in the balanced nodes in these two mechanisms is:

$$P - 2a - P + ka = k(a-2) \quad (11)$$

Then:

$$L_{k_{Trad}} = \sqrt{\frac{\sum_{p=1}^{p-ka}(\overline{W_k} - W_{pk})^2}{P} + \frac{\sum_{p=ka}^{p-2k}(\overline{W_k} - W_{pk})^2}{P}} \quad (12)$$

$$L_{k_{new}} = \sqrt{\frac{\sum_{p=1}^{p-ka}(\overline{W_k} - W_{pk})^2}{P}} \quad (13)$$

$$L_{k_{Trad}} > L_{k_{new}} \rightarrow \frac{L_{k_{new}}}{L_{k_{Trad}}} < 1 \quad (14)$$

With respect to (14), we have:

$$\frac{e_{k_{new}}}{e_{k_{Trad}}} = \frac{2(L_0 - L_{k_{new}})}{L_0 - L_{k_{Trad}}} \Rightarrow \frac{e_{k_{new}}}{e_{k_{Trad}}} > 2 \quad (15)$$

One of the most important parameters in the efficiency of the new mechanism is the ant's memory space (a). In an extreme case, if a=2, then the mechanism resembles the seminal one, with half steps (S), i.e.

$$S_{new} = 1/2 * S_{Trad} \quad (16)$$

Consider that memory space (a) is effective if and only if it can be filled during the ant's wandering steps. Therefore, if a increases, then the amount of steps (S) must increase accordingly to prevent performance degradation. This means that:

$$\text{If } a \rightarrow \infty \text{ then } S \rightarrow \infty \quad (17)$$

Increasing S causes a decrease in load balancing frequency and consequently an increase in convergence time.

Overly long trips also lead to many reserved nodes. At the same time, there may be other roaming ants looking for free, unbalanced nodes. On the other hand, expanding the ant's memory leads to occupying too much bandwidth as well as increasing processing time. Actually, there is a trade-off between the step counts (S) and the memory allocated to each ant (a).

If a<<S, then the memory allocated expires rapidly and the ant is compelled to generate new ants. This explodes the ant population, subsequently augments their communication and the remaining pheromone and finally leads to an increase in time. However, as the probability of balancing every node more than once rises, the load balancing level falls.

On the other side, if a→S, then the probability of creating new ants lessens. Subsequently, the ant's population is reduced. Cutting down on the ant population results in faster speed, diminished communication and the pheromone left by the ants. The final result, however, is not satisfactory (final load balancing level is high). Due to the reasons discussed and with respect to several experiments shown in Figures 4, 5 and Table 1, we deduce that, in order to satisfy the different parameters mentioned, it is better to set the allocated memory at about half of the step counts.

$$a \cong S/2 \quad (18)$$

Experiments achieved with a different memory size allocated, where S=15 initially, are reported here.

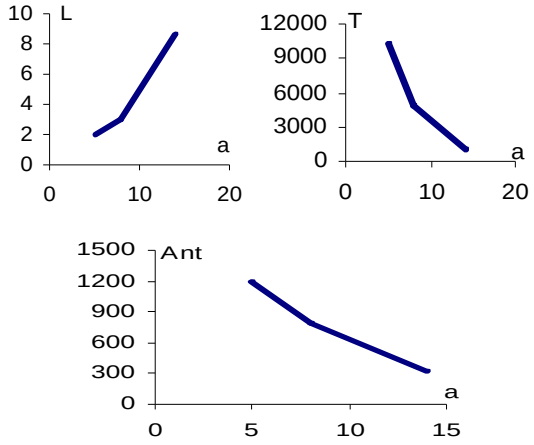

Figure 4: Relation between memory allocated to each ant (a) and a) load balancing level(L) b) time(T) base on millisecond c) Ant no (Ant), where the Ant initial Step Counts (S)=1

a	Time(ms)	Level	Ant No
5	10274	1.9455	1197
8	4906	3.0363	797
14	971	8.6015	325

Table 1: Relation between ants' memory size (a) and Ants with initial Step Counts (S=15).

5.2 Load balancing speed

Adaptively determining the step counts, actually, causes a differentiation in load balancing frequency over time. In other words, as time increases, the whole system approaches convergence and the load balancing frequency lessens, hence postponing the final convergence time. On

the contrary, the new mechanism imposes less overhead as the system nears the balance state. In reality, in an environment such as the grid, attaining final convergence is impractical because of its inherent dynamism and vastness. However, if balancing occurs, it would not last long.

Figure 5 shows a schematic comparison for load balancing frequency between the new and the seminal mechanism.

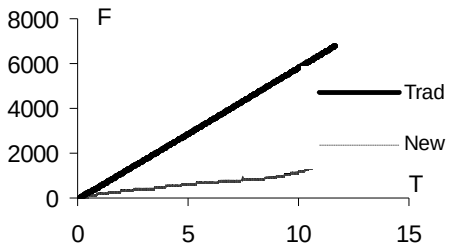

Figure 5: Comparison between the Seminal (Trad) and the new method's load balancing frequency (F).

5.3 Experimental results

Experiments are achieved according to the specifications of Simorgh mini-grid [25]. This mini-grid will include different clusters throughout Ferdowsi University. However, as this mini-grid is under construction now, we have simulated its behaviour. In this simulation, Agent system, Workload, and Resources are modelled as follows:

- Agents. Agents are mapped to a square grid. This simplification has been done in similar works [6, 13]. 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.

The first experiment involves total network connections. In this experiment, as shown in Figure 6.a, ant communication in the new mechanism is drastically less than in the seminal approach. This is mainly because every time an ant wanders 'S' steps in the new method, it balances 'k' elements. In the traditional method, however, the ant wanders '2S+1' steps and then balances only two elements. Therefore, as seen below, with an equal initial step count (S=15), the ant in the new mechanism only goes through 2,000 stages to achieve final convergence, while in the traditional method, the ant passes 7,000 stages. Figure 6.b illustrates the comparison between a colony of ants using S=15 and a memory size=7. This figure elucidates that, in the new mechanism, the communication count goes flat. This occurs when the step counts enlarge and load balancing frequency decreases, i.e. in the last seconds.

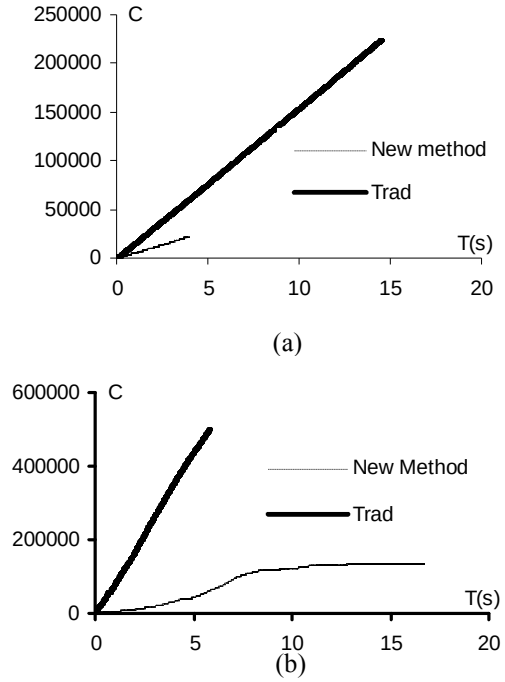

Figure 6: Comparing agent communications (C) between the new and seminal (Trad) method. Final results using. a) One ant S=15, a=7 b) a colony of ants, N=220, S=15, a=7

The second experiment focuses on the relation between load balancing levels and the number of dead ants. As illustrated in Figure 7, as the number of dead ants rises, the load balancing level declines, i.e. it approaches final convergence. This experiment is conducted with different initial ants. Repeating the experiment with a different number of initial ants proves that, deploying more ants would result in better balancing level.

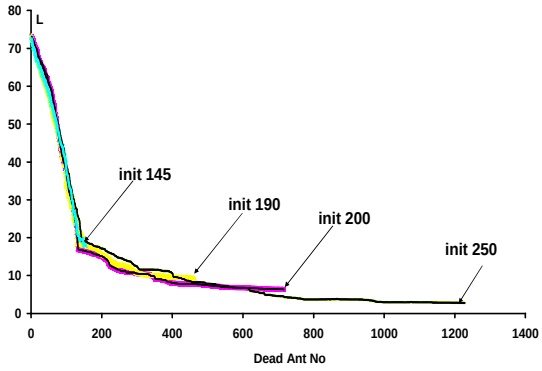

Figure 7: Impact number of created ants (DeadAntNo) on the load balancing level (L), the experiment achieved with different numbers of initial ants (init).

The third experiment concentrates on the correlation between an ant's step counts and the load balancing level. The average step counts of the swarm over time are used for measurement. As Figure 8 shows, the step count increases by approaching convergence. This results in delay to achieve final convergence.

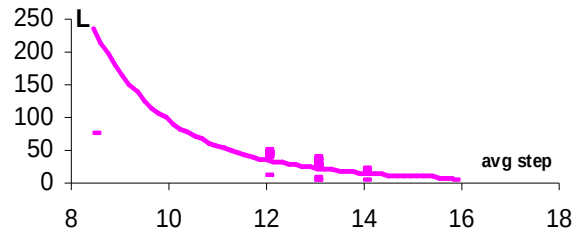

Figure 8: Relation between average Step count (avgStep) and load balancing Level (L)

The fourth experiment indicates the effect of proposed load balancing method on the final job distribution. As understood from Figure 9, ant level load balancing produces a better convergence.

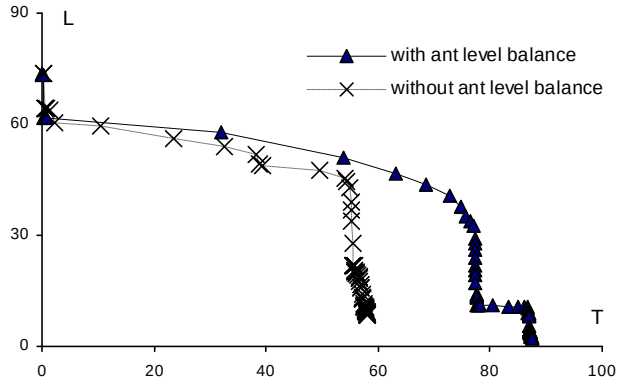

Figure 9: Comparison between effects of using ant level load balancing on final balancing level (L) during the time (T).

It is clear that this load balancing method cannot be achieved without any cost. As illustrated in Figure 9, although the proposed method results are better than previous ones, it consumes more time. We must acknowledge that the new method enables the ant to obtain global information even while moving locally. Furthermore, the validity of the exchanged information is guaranteed in contrast to using the pheromone, which is not, even with evaporation.

The fifth experiment presents the efficiency of the new method in comparison with the seminal approach. Figure 10 illustrates that the new method, with different initial steps and different memory allocated, is more efficient than the seminal one.

On the other hand, comparing the new method's efficiencies, with different initial step counts(S) and different memory allocated shows the effect of the trade-off in determining the memory allocated to each ant (a). In this case, if the memory allocated is high, e.g. a=10, then the probability of creating new ants decreases. So, the probability of visiting a node by different ants is decreased which causes a fall in efficiency. In the other way, as mentioned earlier, low values for memory allocated (a), e.g. a=5, increase the ant population and consequently their interconnections (Ck). This again results in decreasing the final efficiency in regard to (6).

Consider that stages do not have a completely standard meaning in our method. Thus, we think of periods of time as stages (k).

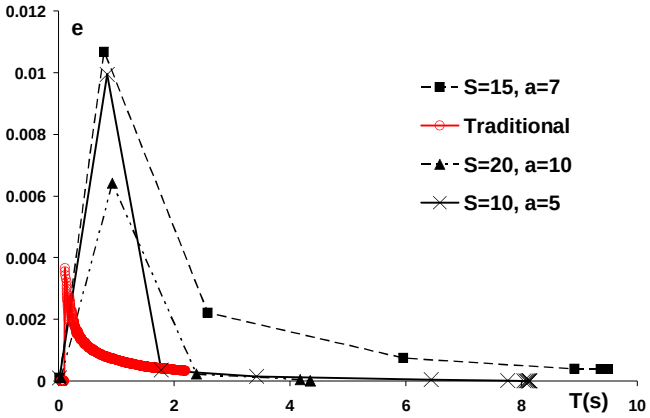

Figure 10: Efficiency (e) comparison between the traditional and the new method with different step counts and memory allocated, during the time (T).

6 Conclusion

As described in the previous sections, equalizing the load of all available resources is one of the most important issues in the grid. In this way, with respect to grid specifications, an echo system of autonomous, rational and adaptive ants is proposed to meet the challenges of load balancing. There are great differences between the proposed mechanism and similar mechanisms which deploy ant colony optimization. We believe that ant level load balancing is the most important difference.

In future work, we plan to extend the applications of ant level load balancing in addition to implementing this mechanism in a more realistic environment. Promoting ant intelligence and adaptation, establishing billing contracts among resources as they exchange customer loads, as well as overcoming security concerns are other future work.

References

[1] F. Berman, Anthony J.G. Hey, Geoffrey C. Fox (2003) *Grid Computing Making the Global Infrastructure a Reality* WILEY SERIES IN COMMUNICATIONS NETWORKING & DISTRIBUTED SYSTEMS.

[2] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd (2002) *ARMS: an Agent-based Resource Management System for Grid Computing*, Scientific Programming, Special Issue on Grid Computing Vol. 10, No. 2 pp. 135-148.

[3] M. Baker, R. Buyya, and D. Laforenza (2002) *Grids and Grid Technologies for Wide-area Distributed Computing*, Software: Practice and Experience Vol. 32, No. 15 pp. 1437-1466.

[4] J. Cao, D. J. Kerbyson, G. R. Nudd (2001) *Performance evaluation of an agent-based resource management infrastructure for grid computing* in

Proc. 1st IEEE Int. Symp. on Cluster Computing and the Grid, pp. 311-318.

[5] J. Cao and F. Zimmermann (2004) *Queue Scheduling and Advance Reservations with COSY* in Proc. of 18th IEEE Int. Parallel and Distributed Processing Symp pp. 120-128.

[6] J. Cao (2004) *Self-Organizing Agents for Grid Load Balancing* Proc. of the 5th IEEE/ACM Int. Workshop on Grid Computing, pp.168-176.

[7] A. Y. Zomaya, and Y. The (2001) *Observations on using genetic algorithms for dynamic load-balancing*, IEEE Trans. on Parallel and Distributed Systems, Vol. 12, No. 9, pp. 899-911.

[8] O. Remien, J. Kramer (1992) *Methodical analysis of adaptive load sharing algorithms*, IEEE Trans. on Parallel and Distributed Systems, Vol. 3, No: 11, pp. 747-760.

[9] Bing Qi Chunhui Zhao (2007) Ant Algorithm Based Load Balancing for Network Sessions, *ICNC 2007*, 3th Int. Conference on Natural Computation, pp. 771-775.

[10] Y. Lan, T. Yu (1995) *A Dynamic Central Scheduler Load-Balancing Mechanism*, Proc. 14th IEEE Conf. on Computers and Communication, Tokyo, Japan, pp. 734-740.

[11] H.C. Lin, C.S. Raghavendra (1992) *A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC),* IEEE Transaction on Software Engineering, Vol. 18, No. 2, pp. 148-158.

[12] Z. Zeng, B. Veeravalli (2004) *Rate-Based and Queue-Based Dynamic Load Balancing Algorithms in Distributed Systems*, *Proc.* 10th IEEE Int. Conf. on Parallel and Distributed Systems, pp. 349- 356.

[13] M. Amini, H. Deldari (2006) *Grid Load Balancing Using an Echo System of Ants*, Proc. Of 24th IASTED Int. Conf, Innsbruck, pp. 47–52.

[14] E. Bonabeau, M. Dorigo, G. Theraulaz (1999) *Swarm Intelligence: from natural to artificial systems*, Oxford University Press, pp: 75-98.

[15] M. T. Islam, P. Thulasiraman, R. K. Thulasiram (2003) *A Parallel Ant Colony Optimization Algorithm for All-Pair Routing in MANETs*, Proc. 3th Int. Symp., Parallel and Distributed Processing, pp. 259-270.

[16] Siriluck Lorpunmanee, Mohd Noor Sap, Abdul Hanan Abdullah, and Chai Chompoo-inwai (2007) *An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment*, Int. Journal of Computer and Information Science and Engineering Volume 1 Number 4, pp. 207-214.

[17] J. Deneubourg, S. Goss, N. Franks (1990) *The dynamics of collective sorting robot-like ants and ant-like robots*, Proc. of the 1st Int. Conf. on Simulation of Adaptive Behavior, pp. 356-363.

[18] Ö. Babaoglu, H. Meling, A. Montresor (2002) *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems*, in Proc. of 22th IEEE Int. Conf. on Distributed Computing Systems, Vienna, Austria, pp. 15-22.

[19] J. Liu, X. Jin, and Y. Wang (2005) *Agent-Based Load Balancing on Homogeneous Minigrids: Macroscopic Modeling and Characterization*, IEEE TRANS. ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL 16, NO 7, pp.586-598.

[20] A. Montresor, H. Meling, and Ö. Babaoglu (2002), Messor: *Load-Balancing through a Swarm of Autonomous Agents*, in Proc. of 1st ACM Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, pp.112-120.

[21] A. Shaout, P. McAuliffe (1998) *Job scheduling using fuzzy load balancing in distributed system*, ELECTRONICS LETTERS, Vol. 34, No. 20, pp. 56-62.

[22] Hai Zhuge, Jie Liu (2004) *A fuzzy collaborative assessment approach for Knowledge Grid*, Int. Journal of Future Generation Computer Systems, Vol. 2, No 20, pp. 101-111.

[23] M. Dorigo, L. Maria (1997) *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*, Transactions ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1, pp. 53-66.

[24] M. Dorigo, G. Carol (1999) *Ant Colony Optimization: A New Meta-Heuristic*, proc. Of 3th Fuzzy Sets and Systems, pp. 21–29.

[25] http://profsite.um.ac.ir/~hpcc