Proceedings of the 24th IASTED International Multi-Conference
PARALLEL AND DISTRIBUTED COMPUTING AND NETWORKS
February 14-16, 2006, Innsbruck, Austria

GRID LOAD BALANCING USING AN ECHO SYSTEM OF INTELLIGENT ANTS

Mohsen Amini Salehi, Hossain Deldari
Department of Software Engineering, Faculty of Engineering Ferdowsi University of Mashhad
Iran
mo_am88@stu_mail.um.ac.ir , hd@um.ac.ir

ABSTRACT
A computational grid is a widespread computing environment that provides huge computational power for large-scale distributed applications. One of the most important issues in such an environment is resource management for which agent-based approaches are appropriate. Load balancing as a part of resource management, has a considerable effect on performance. Ant colony is a metaheuristic that can be instrumental for grid load balancing. This paper presents an echo system of intelligent, autonomous and cooperative ants. The ants in this environment can procreate and also may commit suicide depending on existing conditions. A new concept called *Ant level load balancing* is presented for improving the performance of the mechanism. A performance evaluation model is derived. Theoretical analyses and simulation results indicate that this new mechanism surpasses its predecessor.

KEY WORDS
ARMS, Ant colony, Grid computing, and Load balancing

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 environment should provide access to all available resources seamlessly and fairly.
Resource manager is an important infrastructural component of a grid computing environment. Its overall aim is to efficiently schedule applications that need to utilize the available resources in the grid environment.
ARMS is an agent-based resource manager infrastructure for the grid [1]. In ARMS, each agent can simultaneously stand for a resource questioner, resource provider, and also a matchmaker. Details of the design and implementation of ARMS can be found in [1]. In this work we use ARMS as an experimental platform.
Cosy is a job scheduler that supports job scheduling as well as advanced reservations [2]. It is integrated into ARMS agents to perform global grid management; Cosy needs a load balancer to better utilize available resources.
Load balancing algorithms are designed to spread the load on resources equally and maximize their utilization while minimizing the total task execution time [3]. This is crucial in a computational grid where the most important issue is to fairly assign jobs to resources.
In general, load balancing mechanisms can be broadly categorized as centralized or decentralized, dynamic or static, and periodic or non-periodic. There is a good survey on them in [3].
In [4], Cao et al propose a self-organizing load balancing mechanism using ants, but it was simple and inefficient, we call it the seminal approach. The main contribution of this paper 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. The intelligent ants have memory and learning capabilities. They react to their environment and so better utilize grid resources.
The rest of the paper is organized as follows: Section 2 introduces Ant colony optimization and self-organizing mechanisms for load balancing. Section 3 describes the proposed mechanism. The performance metrics and simulation results are included in Section 4. At the end, we present the conclusion of the article as well as the future works of this research.

2. ANT COLONY OPTIMIZATION

Swarm intelligence [5] is an area inspired by observing the behaviors of insects such as wasps, ants or honey bees. The ants, for example, have little intelligence for their hostile and dynamic environment. Ants perform activities such as organizing their dead into cemeteries and foraging for food. They organize such activities through indirect communication known as “stigmergy”. This communication accomplishes with a chemical substance deposits from ants called pheromone.
Inspiring such ants achievements are used in solving some heuristic problems, like optimal routing, coordinating robots, sorting and especially load balancing [4, 6, and 7]. The main contribution is done in load balancing context by Messor [7].

2.1 Messor

Messor is a grid computing system that is implemented on top of the Anthill framework [7].
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 stages, the ant balances the two overloaded and underloaded nodes found. Once an ant

meets a node, it retains its information about the nodes visited. Other crossover ants can use this information to perform more efficiently. However, with respect to the dynamism of the grid, this information could not be reliable for a long time and may even cause erroneous decision making by other ants.

2.2 Self-Organizing Agents for Grid Load Balancing

Cao proposed a modified mechanism in [4], in which an ant always wanders *'2m+ 1'* steps to finally balance two overloaded and underloaded nodes.

As stated in [4], 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, the ant will initiate the load balancing process frequently. If an ant initiates with a larger *m*, then it wanders for a longer time and decreases the frequency of performing the load balancing process. This implies that the ant's step count should be determined according to the system's load. However, with this method the number of ants and the number of their steps are defined by the user and would not change during the load balancing process. In fact, the user's defining the number of ants and their wandering steps 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 defects, 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.

3. PROPOSED METHOD

In the new mechanism, we propose an echo system of intelligent ants which react commensurately to their conditions. Interactions between these intelligent, autonomous ants result in load balancing throughout the grid.

Here, echo system means that ants are created on demand to achieve load balancing during their lives adaptively. 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,'* and in the end of the *'m'* steps wandering, balances *'k'* overloaded with *'k'* underloaded nodes, instead of one overloaded with one underloaded. This results in an earlier convergence of the system with fewer ants and less communication overhead.

In the next sections, we describe the proposed method in more details.

3.1 Creating Ants

If a node understands that it is overloaded, it can create a new ant with a few steps to balance the load as quickly as possible.

There are many ways in which a node can measure its load (overloaded or not). We consider the load as the number of the jobs waiting in the ready queue of a node, for the sake of simplicity.

3.2 Moving and Deciding

A memory space is allocated to every ant in which the ant records specifications of the environment while 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 neighbors.

3.2.1 Deciding Algorithm

After entering a node, the ant first checks its memory to determine whether this node was already visited by itself or not. If not, the ant can determine the condition of the node, i.e. overloaded, underloaded or 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 perform adaptively by applying 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, 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 it for building membership functions of the node's workload.

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 measure the ant's knowledge about the environment. How this knowledge is used is explained in the next sections.

We can express the inference system as following relation:

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

3.2.2 Ant Level Load Balancing

In the subtle behavior 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 this inspiration.

With respect to the system structure, it is more possible 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, e.g. one has specifications of four overloaded and two underloaded while the other ant has two overloaded and six underloaded in the same position, in this situation, ants can balance their load by exchanging knowledge. We call this “ant level load balancing”. In the last 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 result in better performance in the last step, when the ant wants 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 and this causes better awareness of the environment. There is a similar idea which is inspired from the pheromone deposits from the ants while wandering. Other ants can pursue the ant by using this pheromone deposited. This idea is applied in most of ant colony optimization problems [5]. There is, however, a subtle difference between these two ideas. The information retained by the ant may become invalid over time. This problem can be solved by evaporation [5], however, evaporation is not applicable in some cases, e.g. in the grid, where load information varies frequently. In the new idea, though, the00 knowledge exchanged is completely reliable.

3.2.3 Creating New Ants While Wandering

In special conditions, especially when the ant’s life span is long while the ant is continuing it’s wandering, its memory may get full, but it still encounters nodes which are overloaded or underloaded. In this situation, if a node’s load is overloaded, the ant bears a new one with predefined steps. If the ant encounters an underloaded node, it immediately exchanges its specification with the biggest load in the list of underloaded elements. This results in a better balancing performance and more adaptability to the environment. Here, adaptability translates into increasing the number of the ants automatically, whenever there are many overloaded nodes.

3.3 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 during its wanderings and then disperse the amount of load among them equally. With high probability, the number of elements in the Max list and the Min list are close because of using ant level load balancing, this improves the performance.

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 as stated in previous sections, the ant’s step counts *(m)* must be commensurate to system conditions [4]. 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. To do this requires the ant’s knowledge about the environment. This knowledge should be based 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, and considering 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.

The fuzzy controller 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* for short). In other words, the number of overloaded, underloaded and equilibrium nodes seen during the *LastM*, indicates the recent condition of the environment, while the “*LastM*” itself reports lifetime history of the ant.

This fuzzy system can be stated as a relation as follows:

$$\begin{aligned} &R_B : MaxCount< l,m,h > * MinCount< l,m,h > \\ &* AvgCount< l,m,h > * LastM < TL,L,M,H,TH > \\ &\rightarrow NextM < TL,L,M,H,TH,Dead > \end{aligned} \qquad (2)$$

In this system, a large number of underloaded and especially equilibrium elements indicate equilibrium states. Consequently, the *NextM* should be prolonged, in which case it lowers the load balancing frequency. One can say that, If an ant’s step counts extend to extreme values, its effect tends to be zero. Base on this premise, we can conclude that an ant with too long step counts does not have any influence on the system balance. Rather, it 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.

4. PERFORMANCE EVALUATIONS

In this section, we investigate several common statistics to show the performance of the mechanism described.

4.1 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:

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

$$\overline{W_k} = \frac{\sum_{p=1}^{P} W_{pk}}{P} \qquad (3)$$

The mean square deviation of W_{pk} that describing the load balancing level of the system, is defined as:

$$L_k = \sqrt{\frac{\sum_{p=1}^{P}(\overline{W}_k - W_{pk})^2}{P}} \qquad (4)$$

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

$$e_k = \frac{L_0 - L_k}{C_k} \qquad (5)$$

Where e_k means efficiency at step k and C_k is the total number of agent connections that have been made, to achieve a load balancing level L_k. To compare the efficiency of these two mechanisms, we should consider $e_{k_{new}} / e_{k_{Trad}}$.

As 'L_0' indicates the load balancing level at the beginning of the load balancing process and is equal in both new and seminal mechanisms, we shall discuss the value of $L_{k.}$

For the sake of simplicity, assume that every node gets to $\overline{W}_k$ after balancing process, and needs no more balancing, i.e.

$$\overline{W}_k - W_{pk} = 0 \qquad (6)$$

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

$$L_{k_{new}} = \sqrt{\frac{\sum_{p=1}^{p-ka}(\overline{W}_k - W_{pk})^2}{P}} \qquad (7)$$

While in the seminal approach we have:

$$L_{k_{Trad}} = \sqrt{\frac{\sum_{p=1}^{p-2k}(\overline{W}_k - W_{pk})^2}{P}} \qquad (8)$$

As we suppose that *'a>2'*, we can conclude:

$$P - 2k > P - ka \qquad (9)$$

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

$$P - 2a - P + ka = k(a-2) \qquad (10)$$

Then:

$$L_{k_{Trad}} = \sqrt{\frac{\sum_{p=1}^{p-ka}(\overline{W}_k - W_{pk})^2}{P} + \frac{\sum_{p=ka}^{p-2k}(\overline{W}_k - W_{pk})^2}{P}} \qquad (11)$$

$$L_{k_{new}} = \sqrt{\frac{\sum_{p=1}^{p-ka}(\overline{W}_k - W_{pk})^2}{P}} \qquad (12)$$

$$L_{k_{Trad}} > L_{k_{new}} \rightarrow \frac{L_{k_{new}}}{L_{k_{Trad}}} < 1 \qquad (13)$$

With respect to (13), we have:

$$\frac{e_{k_{new}}}{e_{k_{Trad}}} = \frac{2(L_0 - L_{k_{new}})}{L_0 - L_{k_{Trad}}} \longrightarrow \frac{e_{k_{new}}}{e_{k_{Trad}}} > 2 \qquad (14)$$

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).*

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

$$\text{If } a \to \infty \text{ then } S \to \infty \qquad (15)$$

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

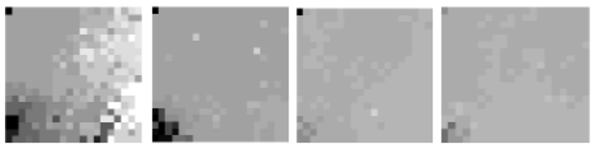
The other side effect of overly trips is reserving many nodes for balancing (by means of remaining pheromone in the nodes) causes them to balance too late; moreover other ants should wander a lot to find a free unbalanced node. These side effects result in performance degradation. On the other hand, increasing memory space results in occupying too much space, it's processing time by the nodes and communication overhead.

Actually there is a trade-off between the step counts *(S)* and memory allocated to each ant *(a)*.

If $a<<S$, then the memory allocated expires rapidly and the ant compels to generate new ants. This explodes the ant population and subsequently increases their communication and also remaining pheromone; finally leading an increase in time. The load balancing level, though, decreases because of increasing the probability of balancing every node more than one time.

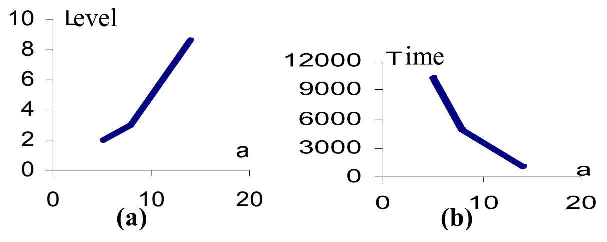
On the other side, if $a \rightarrow S$, then the probability of creating new ants decreases. Subsequently the ant's population reduces. Cutting down the ant population results in increasing speed and decreasing the communication and the pheromone left by the ants. The final result, however, is not satisfactory (final load balancing level is high).Due to the above mentioned reasons and with respect to several experiments shown in Figures 1,2 and Table 1, we deduce that to satisfy the different parameters alluded to, it is better to set allocated memory at about half of the step counts.

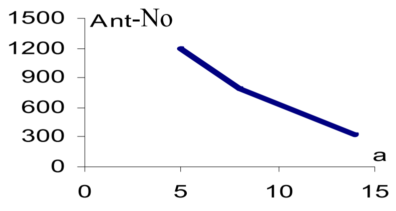
Experiments achieved with different memory size allocated, where *S=15* initially, are reported here.



A) Mem=15 B) Mem=10 C) Mem=7 D) Mem=5

Figure 1. Comparing final convergence with different memory space





(c)

Figure 2. Relation between memory allocated and a) load balancing level b) time c) Ant no, where the Ant initial Step Counts (S)=15

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 memory size (a) and ants with initial Step Counts (S=15)

4.2 Load Balancing Speed

Actually, adaptively determining the step counts 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 decreases, hence postponing the final convergence time. On the contrary, the new mechanism imposes less overhead as the system nears to the balance state. In reality, in an environment such as the grid, attaining final convergence is impractical because of its inherent dynamism/vastness and if balancing occurs, it would not last long.
Figure 3 shows a schematic comparison for load balancing frequency between the new and the seminal mechanism.

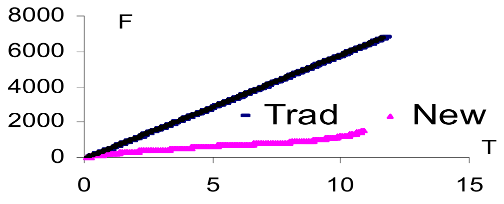
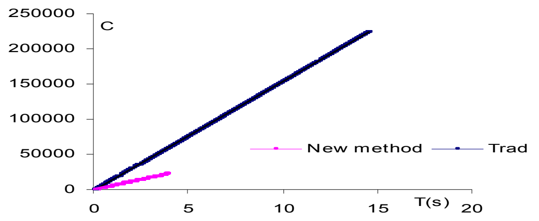


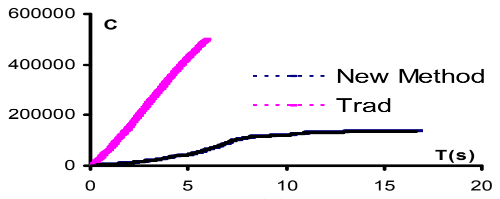
Figure 3. Comparison between the seminal (Trad) and the new method's load balancing frequency (F).

4.3 Simulation and Experimental Results

Simulations are achieved according to the specifications proposed in [4]. The agent system and the mechanism are modeled in a simplified way to outline the system behavior. In this way, the *agents* are mapped to a square grid. The squares used in experiments include all 400 agents. The *Workload* is represented by a value in each agent, which is determined randomly. *Resources* are considered static and all of the agents having a capability value of 100.
The first experiment involves total network connections. In this experiment, as shown in Figure 4.a, ant communication in the new mechanism is drastically less than in the seminal approach. This is because, in the new method, every time an ant wanders *'S'* steps, it balances *'k'* elements. While in the traditional method, 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 2000 stages to get final convergence, while traditional method, passes 7000 stages. Figure 4.b illustrates the comparison between a colony of ants using *S=15* and a memory size=*7*. This figure illustrates 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.



(a)



(b)

Figure 4. 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

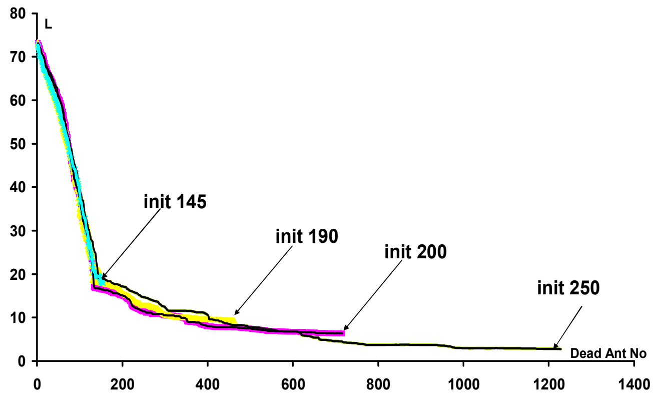


Figure 5. Impact number of created ants on the load balancing level (L), the experiment achieved with different numbers of initial ants (init).

The second experiment focused on the relation between load balancing levels and the number of dead ants.
As can be seen in Figure 5, as the number of dead ants rises, load balancing level decline, i.e. it approaches final convergence. This experiment is conducted with different initial ants. Repeating the experiment with a different initial number of ants proves that, if more ants are deployed, the load balancing level improves.
The third experiment concentrates on the correlation between an ant's step counts and the load balancing level.

We use the average step counts of the swarm over time for measurement. It is obvious from Figure 6 that when approaching convergence, the step count increases. This causes the delay to final convergence.

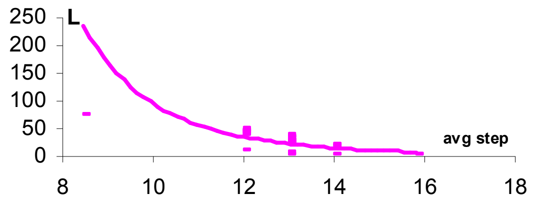


Figure 6. Relation between Step count(S) and the load balancing Level (L).

The fourth experiment indicates the effect of ant level load balancing on the final result. As seen in Figure 7, ant level load balancing produces a better convergence.

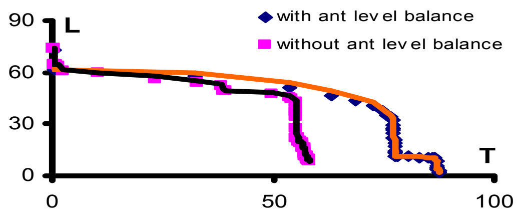


Figure 7. Effects of using Ant level balancing level on Balancing Level (L).

It is clear that ant level load balancing cannot be achieved without costs. As illustrated in Figure 7, ant level load balancing consumes more time, although the results are better. We must acknowledge that this causes the ant to obtain global information even while moving locally.

The fifth experiment presents the efficiency of the new method in comparison with the seminal one. As shown in Figure 8, the new mechanism with different initial step counts is more efficient.

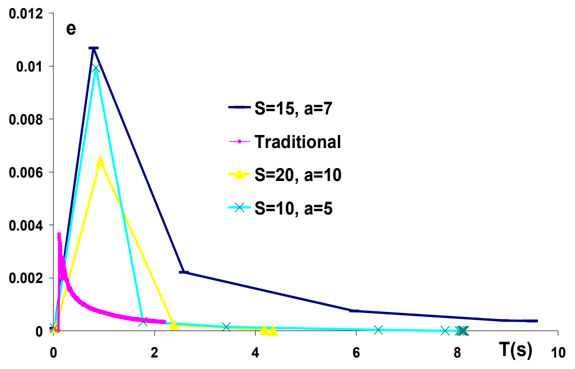


Figure 8.Efficiency (e) comparison between the traditional and the new method with different step counts and memory allocated.

On the other hand, comparing the new mechanism's efficiencies, with different initial step counts, shows the effect of the trade-off in determining the initial step count. In this case, if the initial S is high, e.g. *S=20*, then, as the probability of balancing a node decreases by more than one time, the balancing level *(L_k)* increases, causing a fall in L_0-L_k and, consequently, final efficiency. In the other way, low values for initial *S*, e.g. *S=10*, as mentioned earlier, increases the ant population and consequently their connections *(C_k)*. This again results in decreasing the final efficiency by considering (5).

Consider that the stage has not a completely true meaning in our method. Instead, we think of periods of time as stages *(k)*.

5. 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 was proposed to meet the challenge 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 our future work, we plan to extend the applications of ant level load balancing in addition to implementing the mechanism in a more realistic environment, thus promoting the ant's intelligence and adaptation as well as adding billing contracts between resources as they exchange customer loads and overcome security considerations.

ACKNOWLEDGEMENTS

This research work was supported by Iran Telecommunication Research Center (ITRC). We are also grateful to Mrs. Bahare Mokarram for her helps, Mr. Adel Najjaran Toosi for his ideas and Mr. Saeed Ghani Abadi and Mrs. Adele Amini for their editing.

REFERENCES

[1] J.Cao, *Agent-Based Resource Management System (ARMS)*, PhD Thesis, Warwick University, Dept. of Computer Science, 2001.

[2] ttp://www.ccl-nece.de/macj/software.htm

[3] Y. Zomaya, and Y. Teh, Observations on using genetic algorithms for dynamic load-balancing, *IEEE Trans. on Parallel and Distributed Systems, 12*(9), 2001, 899-911.

[4] J. Cao, Self-Organizing Agents for Grid Load Balancing, *Proc. 5th IEEE/ACM International Workshop on Grid Computing,* 2004, 388-395.

[5] M. Dorigo, G. Caro, Ant Colony Optimization: A New Meta-Heuristic*, IEEE,* 1999.

[6] J. Liu, X. Jin, Y. Wang, Agent-Based Load Balancing on Homogeneous Minigrids: Macroscopic Modeling and Characterization, *IEEE TRANS. ON PARALLEL AND DISTRIBUTED SYSTEMS, 16*(7), 2005, 586-598

[7] Montresor, et al, Messor: Load-Balancing through a Swarm of Autonomous Agents, *Proc. of 1st Int. Workshop on Agents and Peer-to-Peer Computing,* Italy, 2002.