Using Artificial Life Techniques for Distributed Grid Job Scheduling

Azin Moallem and Simone A. Ludwig
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
{a.moallem, ludwig}@cs.usask.ca

ABSTRACT

Grids are an emerging infrastructure providing distributed access to computational and storage resources. Handling many incoming requests at the same time and distributing the workload efficiently is a challenge which load balancing algorithms address. Current load balancing implementations for the Grid are central in nature and therefore prone to the single point of failure problem. This paper introduces two distributed artificial life-inspired load balancing algorithms using Ant Colony Optimization and Particle Swarm Optimization. Distributed load balancing stands out as a robust algorithm in regard to any topology changes in the network. The implementation details are given and evaluation results show the efficiency of the two distributed load balancing algorithms.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Graph and tree search strategies, Heuristic methods, Scheduling.

General Terms

Algorithms, Measurement, Performance, Experimentation.

Keywords

Distributed load balancing, artificial life, ant colony optimization, particle swarm optimization.

1. INTRODUCTION

With the rapid growth of data and computational needs, distributed systems are gaining more attention to solve the problem of large-scale computing [24]. There are several options for establishing distributed systems in which the Grid Systems [17] are the most common ones used for distributed applications [24]. Workload and resource management are two essential functions provided at the service level of the Grid software infrastructure [4]. Resource management and scheduling are key components in addressing the issues of task allocation and load balancing in Grids [23]. In computer networking, load balancing is a technique (usually performed by load balancers) to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to achieve optimal resource utilization, throughput, and response time[1]. The load balancing mechanism aims to spread the load on each computing node equally, maximizing the utilization and minimizing the total task execution time. In order to achieve these goals, the load balancing mechanism should be 'fair' in distributing the load across the computing nodes; this implies that the difference between the "heaviest-loaded" node and the "lightest-loaded" node should be minimized [20].

Methods used in dynamic load balancing can be roughly divided into three classes, i.e., centralized, distributed (decentralized) and hybrid. Each of these classes has its advantages and disadvantages depending on a number of factors, e.g., the size of the system, dynamic behavior, etc. [25]. However, all centralized approaches have the common disadvantage that they need to have global information about the state of the system at each point in time. On the other hand, having all the load balancing information in one node brings up the risk of loosing all the critical data in case of a failure. Accordingly, there has been a great effort in recent years in developing distributed load balancing algorithms while trying to minimize all the communication needs resulting from the distributed nature.

One of the recent trends in designing these distributed dynamic load balancing algorithms involve the use of artificial life techniques. As artificial life techniques have proved to be powerful in optimization problems they are a good candidate for load balancing as we aim to minimize the load between the "heaviest" and "lightest" node. Job scheduling is known to be NP-complete, therefore the use of heuristics is an inevitable approach in order to cope with its difficulties in practice [15].

Among artificial life techniques "social insect systems" are impressive in many ways, but two aspects are of particular interest. First, they are outstandingly robust. They function smoothly even though the colony may be continuously growing, or may suffer a sudden reduction in numbers through an accident, predation, or experimental manipulation, or may spontaneously split into two distinct colonies of half the size [14]. They routinely cope with gross and

[1]http://www.answers.com/topic/load-balancing-computing

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

minor disturbances of habitat and with seasonal variations in food supply [21]. Second, they are tiny insects with no or very small memory and computational ability yet they are surviving in our complex real world because of their large numbers. Robustness gives us the ability to deal with the dynamic topology of today's networks as nodes may come and go arbitrarily; and being "simple" provides us with the efficiency we need in dealing with large scale systems.

Another artificial life technique which performs well in optimization problems is Particle Swarm Optimization (PSO). PSO is a stochastic search method that was developed by Kennedy and Eberhart in 1995 [13]. The algorithm is an evolutionary algorithm (EA) that imitates the sociological behavior of a flock of birds or a group of people. In bird flocking, the population benefits from sharing of information of each individual's discovery and past experience within the whole population. Each individual (called particle) in the population (called swarm) will "fly" over the search space to search for the global minima [18]. PSO is easily implemented as it uses numerical encoding and it is proved to have fast convergence speed [16].

This research proposes and compares two new approaches for distributed load balancing inspired by Ant Colony and PSO. In the Ant Colony approach each job submitted invokes an ant and the ant searches through the network to find the best node to deliver the job to. Ants leave information related to the nodes they have seen as a pheromone in each node which helps other ants to find their paths more efficiently. In the particle swarm approach each node in the network is considered to be a particle and it tries to optimize its load locally by sending or receiving jobs from its neighbors. This process being done locally for each node, results in a partially global optimum of the load within the entire network given time constraint limitations.

The rest of this paper is organized as follow: An overview of related work done in similar areas is provided in Section 2. Brief descriptions of the Ant Colony and PSO techniques are provided in Section 3. The proposed load balancing algorithms are described in Section 4. Section 5 shows experimental results to demonstrate how good the algorithms work and finally Section 6 and 7 are dedicated to conclusion and future endeavors of this research.

2. RELATED WORK

There is a lot of research done in the area of centralized load balancing. As many approaches are out of the scope of this research, we only provide a brief overview of relevant ones.

Subrata et al. [23] used Genetic Algorithms and Tabu search for performing centralized load balancing simulations. They have proved that the two techniques work quite well compared to some classical algorithms like Min-min, Max-min and Sufferage in terms of time makespan. In [6] an agent-based load balancing algorithm is proposed and is applied to drug discovery and design. The algorithm performs well in meeting QoS (Quality of Service) parameters, however as there is a global information repository which maintains the global information of all the resources in the Grid the same problem as in all centralized approaches exist, which is the single point of failure. BDII [1] is another centralized information system which provides the most recent information about the Grid resources to the users. However being a centralized unit with huge computational needs and frequent updates, there still exists a single point of failure in the system.

Research in the area of distributed load balancing is diverse. One of the investigations, similar to the algorithm we propose in this paper, is Messor [19]. Montresor et al. have used an ant colony approach to develop a framework called Anthill which provides an environment for designing and implementing Peer-to-Peer systems. They have developed Messor which is a distributed load balancing application based on Anthill and they have performed some simulations to show how well Messor works. However, they have not addressed the problem of topology changes in the network and do not provide evidence comparing their approach with other approaches in distributed load balancing. In [3] a very similar approach to Messor and also our research is provided for load balancing and different performance optimization strategies are carried out. However, they do not compare their results with other distributed load balancing strategies.

Another related and similar research is done by Al-Dahoud et al. in which each node sends a colored colony throughout the network [2]. This approach helps in preventing ants of the same nest from following the same route and hence enforcing distribution over the nodes in the network. However, the research does not provide any evidence whether this approach has improved load balancing efficiency.

Cao [5] proposes an agent-based load balancing approach in which an agent-based Grid management infrastructure is coupled with a performance-driven task scheduler that has been developed for local Grid load balancing. This work addresses the problem of load balancing in a global Grid environment. A genetic algorithm based scheduler has been developed for fine-grained load balancing at the local level. This is then coupled with an agent-based mechanism that is applied to balance the load at a higher level. Agents cooperate with each other to balance workload in the global Grid environment using service advertisement and discovery mechanisms [5]. In their research, the scalability is of great importance as they are using a genetic algorithm approach in the local level load balancing part. However, the genetic algorithm may be a bottleneck for the system at that stage.

Other similar research, which have benefited from the Ant colony's power, mostly focus on load balancing in routing problems and are introduced in [21, 22]. In [22] the research provides a survey of four different routing algorithms: ABC, AntNet, ASGA and SynthECA and MACO. In [21], the ant colony approach in telecommunication networks is used.

Some researchers have considered job migration in their load balancing algorithms. However, job migration is far from trivial in practice. It involves collecting all system states (e.g. virtual Memory image, process control blocks, unread I/O buffer, data pointers, timers etc.) of the job which is large and complex. Several studies have shown that: (1) job migration is often difficult to achieve in practice, (2) the operation is generally expensive in most systems, and (3) there are no significant benefits of such a mechanism over those offered by non-migratory counterparts. These are the reasons why we are not concerned with job migrations for our approach [23].

Literature using PSO for distributed load balancing is less rich compared to Ant Colony load balancing. One similar

[1]https://twiki.cern.ch/twiki//bin/view/EGEE/BDII

research to what we propose here is done in [1], where a fuzzy based PSO approach is proposed. A position matrix indicates a fuzzy potential scheduling solution. However, the approach is a central approach and does not take into account the arrival of new jobs in a distributed architecture.

3. OPTIMIZATION APPROACHES

3.1 Ant Colony Optimization

Some social insect systems in nature can present an intelligent collective behavior although they are composed of simple individuals with limited capabilities. The intelligent solutions to problems naturally emerge from the self-organization and indirect communication of these individuals. These systems, such as the Ant Colony, provide important techniques that can be used in the development of distributed artificial intelligent systems.

In the early 1990s, Ant Colony Optimization (ACO) [9, 11, 12] was introduced by Dorigo et al. as a novel nature-inspired meta-heuristic for the solution of hard combinatorial optimization (CO) problems [10]. The inspiring source of ACO is the foraging behavior of real ants. Ants use a signaling communication system based on the deposition of pheromone over the path they follow, marking their trail. Pheromone is a hormone produced by ants that establishes a sort of indirect communication among them. Basically, an isolated ant moves at random, but when it finds a pheromone trail, there is a high probability that this ant will decide to follow the trail[1].

A single ant has no global knowledge about the task it is performing. The ant's actions are based on local decisions and are usually unpredictable. The intelligent behavior naturally emerges as a consequence of the self-organization and indirect communication between the ants. This is usually called Emergent Behavior or Emergent Intelligence. Indirect communication between the ants via pheromone trails enables them to find the shortest paths between their nest and food sources[1]. This characteristic of real ant colonies is exploited in artificial ant colonies in order to solve CO problems [8].

What makes the ant colony approach especially interesting for the distributed load balancing problem is the distributed nature. Other artificial life techniques like genetic algorithms, tabu search or the classical PSO, though being quite powerful for optimization problems, they have one drawback when being used in distributed environments; the solutions found by these algorithms should be compared with each other to be evaluated based on their usefulness (i.e. fitness) which in turn however prevents a completely distributed approach. On the other hand ants begin to move toward the optimized solution by communicating indirectly through the environment with other local ants and this communication brings convergence with increasing number of iterations.

Besides the ability of indirect communication via leaving pheromone trails, ants are capable of other complex behaviors without any intelligence incorporated in them. One of them is the ability of ants to cluster objects (like dead corpses) in their nests. At the first glance they may seem to be directed by a leader to cluster objects. However, the ants exhibit a very simple behavior which enables them to cluster objects without any intelligence. Figure 1 shows a flow chart of an ant which moves around randomly until it encounters an object; if the ant is carrying an object it will drop the object, otherwise it picks it up and continues on its way [19].

[1]http://ai-depot.com/CollectiveIntelligence/Ant.html

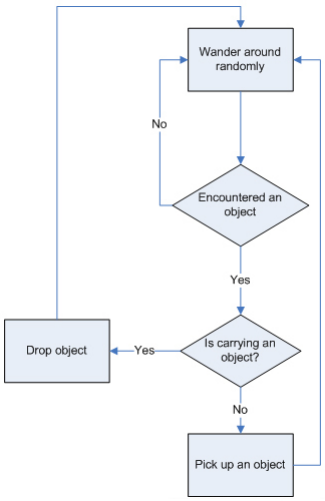

Figure 1: The flow chart of an ant behavior capable of clustering objects.

3.2 Particle Swarm Optimization

One of the advantages of the PSO technique over other social behavior-inspired techniques is its implementation simplicity. Yet, as it is a new technique, it has not been widely used and in particular it is not used for dynamic distributed Grid job scheduling.

In a PSO system, multiple candidate solutions coexist and collaborate simultaneously. Each solution candidate, called a 'particle', flies in the problem search space (similar to the search process for food of a bird swarm) looking for the optimal position to land. A particle, as time passes through its quest, adjusts its position according to its own 'experience' as well as according to the experience of neighboring particles [7].

Two factors characterize the status of a particle in the search space: its position and velocity. The m-dimensional position for the ith particle in the kth iteration can be denoted as $x_i(k) = x_{i1}(k), x_{i2}(k), ..., x_{im}(k)$.

Similarly, the velocity (i.e., distance change) is also an m-dimensional vector which for the ith particle in the kth iteration can be described as $v_i(k) = v_{i1}(k), v_{i2}(k), ..., v_{im}(k)$.

The particle updating mechanism for a flying particle flying can be formulated as Equation 1 and 2:

$$v_{id}^{k+1} = w*v_{id}^{k}+c_1*rand_1*[pbest-x_{id}^{k}]+c_2*rand_2*[gbest-x_{id}^{k}] \quad (1)$$

$$x_{id}^{k+1} = x_{id}^{k} * v_{id}^{k+1} \quad (2)$$

In which v_{id}^{k}, called the velocity for particle i in the kth iteration, represents the distance to be traveled by this particle from its current position, x_{id}^{k} represents the particle position in the kth iteration, $pbest$ represents its best previous position (i.e. its experience), and $gbest$ represents the

NodeIp	Load
192. 168. 35. 25	0.8
...	...

Table 1: Load table information in nodes

best position among all particles in the population. $rand_1$ and $rand_2$ are two random functions with a range [0,1]. c_1 and c_2 are positive constant parameters called acceleration coefficients (which control the maximum step size the particle can achieve). The inertia weight w, is a user-specified parameter that controls the impact of previous historical values of particle velocities on its current one.

A larger inertia weight pressures towards global exploration while a smaller inertia weight pressures toward fine-tuning the current search area. Suitable selection of the inertia weight and acceleration coefficients can provide a balance between the global and the local search. Equation 1 is used to calculate the particle's new velocity according to its previous velocity and to the distances of its current position from both its own best historical position and its neighbors' best position. Then the particle flies toward a new position according to Equation 2 [7].

4. PROPOSED LOAD BALANCING APPROACHES

4.1 Ant Colony Load Balancing

A new load balancing algorithm was developed by merging the idea of how ants cluster objects with their ability to leave trails on their paths so that it can be a guide for other ants passing their way. In the proposed algorithm, when a job is submitted to the Grid from any node in the network, an ant is invoked to search for the best node (i.e. the lightest node) to deliver the job to. As the ant is moving from one node in the network to another, it collects statistical information about the load of the nodes it has visited and carries this information from node to node. Each ant leaves this statistical information in each node as a trail so that other ants can use this information in order to decide which path to take to reach to "lighter" regions in the network.

Ants build up a table in each node as shown in Table 2. This table acts like pheromone an ant leaves while it is moving and guides other ants to choose better paths rather than wandering randomly in the network. Looking at the information provided in these tables ants tend to go toward nodes which are less loaded. Entries of the table are the nodes that ants have visited on their way to deliver their jobs and their load information.

As the number of jobs submitted to the network increases, the ants can take up a huge amount of bandwidth of the network, so moving ants should be as simple and small-sized as possible. To do this instead of carrying the job while the ant is searching for a "light" node, it carries the source node information and a unique job id. By doing this, whenever an ant reaches its destination the job can be downloaded from the source when needed. Results can also be uploaded to the source node directly.

The pseudo-code of a "typical" ant behavior in its initialization and running phases can be seen in Algorithms 4.1 and 4.2. Some details are not included in the pseudo codes to simplify and stress on the main parts.

As shown in Algorithm 4.1, when a job is sent to a node an ant is initialized. The ant first chooses its destination node according to the information provided in the local load balancing table built by other ants. The ant pays attention that the chosen node should be a valid node (an invalid node is a node that existed sometime in the network but it is not there anymore according to a failure).

Algorithm 4.1: INITIALIZE()

$destNode \leftarrow find\_destination\_node(load\_table)$
while $!valid(destNode)$
do
$remove\_from\_localtable(destNode)$
$destNode \leftarrow find\_destination\_node(load\_table)$
$run()$

As the topology of the network may change, the ant updates the load balancing table and repeats this process until it finds a valid node. Thus, the dynamic topology problem is addressed as an issue of self organization inside the algorithm. The ant proceeds to the next step when a valid node is found, moving toward the destination. Details of this phase are shown in the pseudo code given in the Algorithm 4.2. In this phase, the ant moves toward the destination node and while it is going from node to node it adds the information of visited nodes to its own history (*updareHistory()*) and also updates local tables in the nodes on its path (*updateLocalTable(currentNode)*). To improve performance, in a random fashion, the ant chooses one of the neighbors on its path instead of its predefined destination in case the neighbor is "lighter" than the destination node.

Algorithm 4.2: RUN()

$sourceNode, destNode$
if $step! = maximum\_number\_of\_steps$
then $updateHistory()$; $updateLocaltable(currentNode)$; $destNode \leftarrow getHistoryORNeighbourhood()$; $step++$
else $submitjob(destNode)$

4.2 Particle Swarm Load Balancing

We propose a new approach for scheduling jobs in the Grid using the idea of PSO described in Section 3.2. In this approach all the nodes in the Grid are considered a flock and each node is a particle in this flock. Thus, the position of each node in the flock can be determined by its load. The velocity of each position can be defined by the load difference the node has compared to its other neighbor nodes and the particle moves toward the best neighbor by sending some of the assigned jobs to it. In case of a tie no jobs will be exchanged. As mentioned in Section 3.2, the velocity of each particle can be formulated as in Equation 1.

As we are dealing with a dynamic environment, the nature of the problem being solved is changing with time. We do not want to rely on previous history of the solutions found as

in a classical PSO algorithm; therefore in our approach we propose w and c_2 to have the value 0 and c_1 to have the value 1. As the environment of the search space in a dynamic Grid is dynamically changing; using *gbest* will help the particle to move toward its current best neighbors, which is the benefit to be used for load balancing. Thus, the velocity will be the difference between the loads of two nodes. As the load difference between nodes increases, the velocity also increases. Thus, more jobs will be exchanged accordingly in order to bring the Grid to an equilibrium more quickly. The algorithm selforganizes in dealing with topology changes as it sends messages to its current neighbors and makes decisions in real time.

Taking into account that all nodes are exchanging their loads in parallel, will bring the network to a partial global optima very quickly. Each node will submit some of its jobs to one of its neighbors which has the minimum load among all. If all its neighbors are busier than the node itself, no job is submitted by the current node.

The experience of each particle which helps in moving it toward the best solution as in a classical approach was left out deliberately. The reason is the dynamicity of job arrivals in the Grid which makes each particle's past experience useless. Algorithm 4.3 shows the pseudo code for PSO which runs in parallel in all nodes.

Algorithm 4.3: RUN()

$minNode \leftarrow find\_minimumLoadNode\_inNeighbour()$
$delta \leftarrow (currentLoad - minNodeLoad)/2$
if $delta > 0$
then { **for** $i \leftarrow delta$ **to** 0
$minNode.submitJob()$

5. EXPERIMENTAL RESULTS

By providing some experimental results we will compare the performance of the Ant colony and PSO algorithms with the random approach. In the first set of experiments the makespan and the communication overhead is measured and compared. The fairness of the algorithms, their average variance and scalability are measured in the second set of experiments. In order to build the topology of the network with a specific number of nodes we first build a minimum spanning tree out of a specified number of nodes and then by adding random links to the tree we get our final topology. As the network and its configuration are created randomly (although they are roughly the same as their construction method is the same) the same experiment can not be repeated twice, therefore we have used the average of ten runs in our experiments. Each machine has one processor with the speed defined in MHZ and each job has a length which is defined in millions of instructions. Thus, the time each job will take to run on a processor can be forecasted. For the implementation of a scheduling algorithm inside each resource, a First Come First Served (FCFS) algorithm is used.

In the first set of experiments we compare the Ant-Coloy algorithm's makespan with both the PSO and the random approach. The random approach assigns the jobs randomly within the network. The makespan is defined as the amount of time taken from when the first job is sent to the network

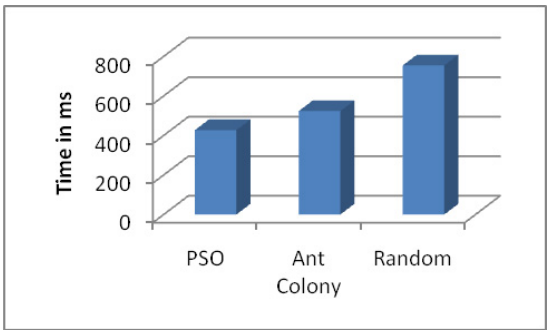

Figure 2: Average makespan of different approaches.

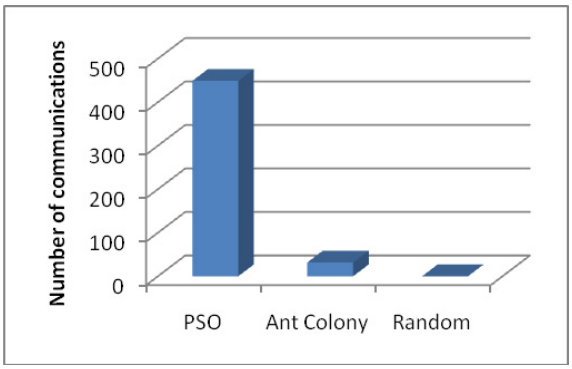

Figure 3: Average number of communications of different approaches.

until the last job gets returned. Figure 2, shows a one hundred node network in which 400 jobs with random lengths are sent to random locations in the network. The average makespan of different approaches are shown in this figure.

Figure 3, plots the number of communications needed for each approach to fulfill the task of job scheduling. In PSO each node tries to communicate with its neighbors to balance its load by submitting or receiving jobs.

As can be concluded from Figures 2 and 3, the random approach performs really poorly in terms of makespan, however, it does not have any communication overhead in the network. PSO and Ant colony perform nearly the same comparing their makespan, but PSO always wins over the Ant Colony approach, however, it uses a huge amount of bandwidth in the network as a job is transferred from one machine to another several times until it gets executed. It is not necessary to send the job several times as it uses up a huge bandwidth; we can simply send a unique job ID to the source node and when the job has reached its final destination and is ready to get executed it can be downloaded from the source node.

A good load balancing algorithm is one which is "fair". A fair algorithm is an algorithm for which the difference between the "heaviest" and "lightest" node is minimized. We are experimenting with a 100 node network and jobs are sent to the network at random time intervals. Figure 4, shows the average load difference between the "heaviest" and "lightest" node for the different approaches. As shown in the figure, PSO wins over other approaches in terms of the fairness measure, meaning that the difference between

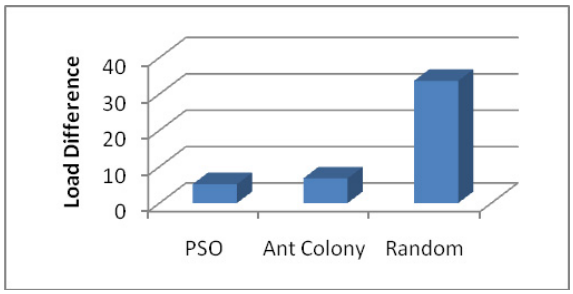

Figure 4: Load difference between heaviest and lightest node of different approaches.

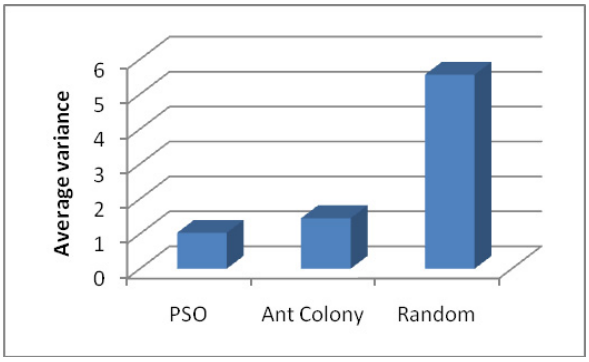

Figure 5: Average variance for different approaches.

the "heaviest" and "lightest" node is minimized in PSO. Ant Colony is performing slightly worse than PSO but still much better than the random approach.

On the other hand a good load balancing algorithm is one in which the load related to the nodes is equally distributed. The performance of the algorithm can be measured by calculating the deviation from the average load for all the nodes. This can be done by calculating the variance given in Equation 3. Comparing each algorithm's deviation with other algorithms is a measure of how fair the algorithm is.

$$variance = \sqrt{\frac{\sum_{i=1}^{n}(\overline{load} - load_i)^2}{n}} \quad (3)$$

As can be seen in Figure 5, for PSO all nodes are near the expected average load node at a given timestamp in the network. Ant colony is also performing fairly good. We would assume all nodes in the network have an equal computational ability; however, different computational abilities do not affect the performance of the algorithm. We would need to leave more information in load tables being built in nodes and hence decided against it.

The scalability of any algorithm is of great importance. Our Ant-colony approach is robust to scaling as a new ant is invoked in response to each job submitted to the network. One of the advantages of this approach in which an ant is invoked in response to submitting a job is that these ants update the load balancing table in each node and hence improve the performance of the system by updating it more often when their number is large. In order to understand how well the Ant Colony algorithm responds to larger scenarios we have investigated the effect of increasing the number of jobs.

Figure 6: Effect of increasing number of jobs on time.

Approach	Equation
Random	Y = 115x + 260.5 = [ms]
PSO	Y = 72.82x + 138.6 = [ms]
Ant Colony	Y = 86.03x + 174.5 = [ms]

Table 2: Load table information in nodes

Figure 6, shows how an increase in the number of jobs will affect the makespan. The horizontal axis shows the number of jobs sent to the Grid and the vertical axis shows the time required to run the load balancing algorithm. As can be seen for PSO and Ant colony the makespan increases linearly while the number of jobs increases. However, the gradient of PSO is slightly less than the gradient of the Ant Colony.

Table 2, lists the regression equations from the measurements shown in Figure 6. PSO scales best with an increasing number of jobs.

In another set of experiments we examined the effect of the number of injection points on the performance of the algorithms. The random approach, as expected, shows a very bad performance when all jobs are sent to one node in the Grid. It performs by a factor of 16.35 worse considering the makespan. Having a mutation factor for the Ant colony approach enables it to perform as good as random injection points. A mutation factor makes an ant to decide randomly with a predefined probability. The particle swarm performs worse by a factor of 3.1 when having only one injection point in the system.

6. CONCLUSION

In this research we investigated the use of artificial life techniques and more specifically social insect systems and sociological behavior of flocks of birds in designing two new distributed load balancing algorithms for the Grid. The simulations show that the algorithms proposed can perform well for job scheduling in a Grid network where jobs are being submitted from different sources and at different time intervals. PSO shows better results considering the makespan, however; it uses up more bandwidth as more communication is needed than for the Ant Colony. Looking at the scalability of the algorithms it shows a linear behavior with an increasing number of jobs. One drawback of the Ant Colony algorithm is that it does not schedule the jobs well in scenarios where all the jobs are being sent to one or two

nodes in the network. This can be fixed by including a mutation factor and the ants move randomly with a certain probability. On the other hand in a normal Grid system in which jobs are submitted randomly from many nodes, increasing the number of jobs, which results in an increase in the number of ants, can be an advantage for the algorithm as the load balancing tables in each node get updated more frequently.

7. FUTURE WORK

Future research will address the following. Node failures or congested nodes need to be considered imposing that the ants have to update/change the tables while delivering their jobs to specific nodes. Furthermore, if load tables in each node do not get updated frequently their information may be outdated which can be considered as pheromone evaporation in Ant colony. However, the idea needs a deeper analysis. Moreover, PSO nodes may want to create new connections in response to losing some of their connections according to node failures. In addition, investigating the effect of node failures on the performance of the algorithms also needs to be investigated.

8. REFERENCES

[1] A. Abraham, H. Liu, W. Zhang, and T. Chang. Scheduling jobs on computational grids using fuzzy particle swarm algorithm. In *Proceedings of 10th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 500–507, 2006.

[2] A. Al-Dahoud and M. Belal. Multiple ant colonies for load balancing in distributed systems. In *Proceedings of The first International Conference on ICT and Accessibility*, 2007.

[3] J. Cao. Self-organizing agents for grid load balancing. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 388–395, Washington, DC, USA, 2004.

[4] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd. Grid load balancing using intelligent agents. *Future Gener. Comput. Syst.*, 21(1):135–149, 2005.

[5] J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd. Agent-based grid load balancing using performance-driven task scheduling. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 49.2, Washington, DC, USA, 2003. IEEE Computer Society.

[6] S. Chen, W. Zhang, F. Ma, J. Shen, and M. Li. A novel agent-based load balancing algorithm for grid computing. In *GCC Workshops*, pages 156–163, 2004.

[7] T. Chen, B. Zhang, X. Hao, and Y. Dai. Task scheduling in grid based on particle swarm optimization. In *ISPDC '06: Proceedings of the Fifth International Symposium on Parallel and Distributed Computing*, pages 238–245, Washington, DC, USA, 2006. IEEE Computer Society.

[8] J. L. Deneubourg, S. Aron, S. Goss, , and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[9] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[10] M. Dorigo and C. Blum. Ant colony optimization theory: a survey. *Theor. Comput. Sci.*, 344(2-3):243–278, 2005.

[11] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical report, 1991.

[12] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:29–41, 1996.

[13] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science, MHS '95.*, pages 39–43, Oct 1995.

[14] N. R. Franks. Army ants: A collective intelligence. *American Scientist*, pages 139–145, March 1989.

[15] C. Grosan, A. Abraham, and B. Helvik. Multiobjective evolutionary algorithms for scheduling jobs on computational grids. ADIS International Conference, Applied Computing 2007, Salamanca, Spain, Nuno Guimares and Pedro Isaias (Eds.), 2007.

[16] J. K. J and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[17] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[18] D. Liu, K. Tan, and W. Ho. A distributed co-evolutionary particle swarm optimization algorithm. *IEEE Congress on Evolutionary Computation, CEC 2007.*, pages 3831–3838, Sept. 2007.

[19] A. Montresor and H. Meling. Messor: Load-balancing through a swarm of autonomous agents. In *In Proceedings of the 1st Workshop on Agent and Peer-to-Peer Systems*, 2002.

[20] S. Salleh and A. Y. Zomaya. *Scheduling in Parallel Computing Systems: Fuzzy and Annealing Techniques*. The Springer International Series in Engineering and Computer Science, 1999.

[21] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 209–216, New York, NY, USA, 1997. ACM.

[22] K. M. Sim and W. H. Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 33(5):560–572, Sept. 2003.

[23] R. Subrata, A. Y. Zomaya, and B. Landfeldt. Artificial life techniques for load balancing in computational grids. *J. Comput. Syst. Sci.*, 73(8):1176–1190, 2007.

[24] K. Q. Yan, S. C. Wang, C. P. Chang, and J. S. Lin. A hybrid load balancing policy underlying grid computing environment. *Comput. Stand. Interfaces*, 29(2):161–173, 2007.

[25] W. Zhu, C. Sun, and C. Shieh. Comparing the performance differences between centralized load balancing methods. *IEEE International Conference on Systems, Man, and Cybernetics, 1996.*, 3:1830–1835 vol.3, Oct 1996.