Process Scheduling Using Ant Colony Optimization Techniques*

Bruno Rodrigues Nery[1], Rodrigo Fernandes de Mello[1],
André Carlos Ponce de Leon Ferreira de Carvalho[1], and Laurence Tianruo Yang[2]

[1] Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação – São Carlos, SP, Brazil
[2] St. Francis Xavier University, Antigonish, NS, Canada
lyang@stfx.ca

Abstract. The growing availability of low cost microprocessors and the evolution of computing networks have enabled the construction of sophisticated distributed systems. The computing capacity of these systems motivated the adoption of clusters to build high performance solutions. The improvement of the process scheduling over clusters originated several proposals of scheduling and load balancing algorithms. These proposals have motivated this work, which defines, evaluates and implements a new load balancing algorithm for heterogeneous capacity clusters. This algorithm, named Ant Scheduler, uses concepts of ant colonies for the development of optimization solutions. Experimental results obtained in the comparison of Ant Scheduler with other approaches investigated in the literature show its ability to minimize process mean response times, improving the performance.

1 Introduction

The growing availability of low cost microprocessors and the evolution of computing networks have enabled the construction of sophisticated distributed systems. In such systems, the processes executed on network computers communicate to each other to perform a collaborative computing task. A load balancing algorithm is frequently adopted to distribute the processes among computers.

A load balancing algorithm is responsible to equally distribute the processes load among the computers of an distributed environment [1]. Krueger and Livny [2] demonstrate that these algorithms can reduce the mean and standard deviation of processes' response times. Shorter response times result in higher performance in the execution of the processes.

The load balancing algorithms involve four policies: transference, selection, location and information [1]. The transference policy determines whether a computer is in a suitable state to participate in a task transfer, either as a server or as a receiver of processes. The selection policy defines the processes that should be transferred from the busiest computer to the idlest one. The location policy is responsible to find a suitable transfer partner (sender or receiver) for a computer, once the transfer policy has decided about

* Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 304–316, 2006.
© Springer-Verlag Berlin Heidelberg 2006

its state. A serving computer distributes the processes, when it is overloaded; a receiving computer requests processes, when it is idle. The information policy defines when and how the information regarding the computers' availability is updated in the system. Several works related to load balancing can be found in the literature [3,4,1,5,6].

Zhou and Ferrari [3] evaluated five server-initiated load balancing algorithms, i.e. initiated by the most overloaded computer: Disted, Global, Central, Random and Lowest. In Disted, when a computer suffers any modification in its load, it emits messages to the other computers to inform its current load. In Global, one computer centralizes all the computers' load information of the environment and sends broadcast messages in order to keep the other computers updated. In Central, as in Global, a central computer receives all the load information related to the system; however, it does not update the other computers with this information. This central computer decides the resources allocation in the environment. In Random, no information about the environment load is handled. Now, a computer is selected by random in order to receive a process to be initiated. In Lowest, the load information is sent when demanded. When a computer starts a process, it requests information and analyzes the loads of a small set of computers and submit the processes to the idlest one, the computer with the shortest process queue.

Theimer and Lantz [4] implemented algorithms similar to Central, Disted and Lowest. They analyzed these algorithms for systems composed of a larger number of computers (about 70). For the Disted and Lowest algorithms, a few process receiver and sender groups were created. The communication within these groups was handled by using a multicast protocol, in order to minimize the message exchange among the computers. Computers with load lower than a inferior limit participate of the process receiver group, whilst those with load higher than a superior limit participate of the process sender group.

Theimer and Lantz recommend decentralized algorithms, such as Lowest and Disted, as they do not generate single points of failure, as Central does. Central presents the highest performance for small and medium size networks, but its performance declines in larger environments. They concluded that algorithms like Lowest work with the probability of a computer being idle [4]. They assume system homogeneity, as they use the size of the CPU's waiting queue as the load index. The process behavior is not analyzed; therefore, the actual load of each computer is not measured.

Shivaratri, Krueger and Singhal [1] analyzed the server-initiated, receiver-initiated, symmetrically initiated, adaptive symmetrically initiated and stable symmetrically initiated algorithms. In their studies, the length of the process waiting queue at the CPU was considered as the load index. This measure was chosen because it is simple and, therefore, can be obtained with fewer resources. They concluded that the receiver-initiated algorithms present a higher performance than the server-initiated ones. In their conclusions, the algorithm with the highest final performance was the stable symmetrically initiated. This algorithm preserves the history of the load information exchanged in the system and takes actions to transfer the processes by using this information.

Mello *et al.* [5] proposed a load balancing algorithm for distributing processes on heterogeneous capacity computers. This algorithm, named TLBA (Tree Load Balancing

Algorithm), organizes computers in a virtual tree topology and starts delivering processes from the root to the leaves. In their experiments, this algorithm presented a very good performance, with low mean response time.

Senger *et al.* [6] proposed *GAS*, a genetic scheduling algorithm which uses information regarding the capacity of the processing elements, applications' communication and processing load, in order to allocate resources on heterogeneous and distributed environments. *GAS* uses Genetic Algorithms to find out the most appropriate computing resource subset to support applications.

These works, together with the development of new computing techniques based on biology, motivated the proposal of a new load balancing algorithm, named Ant Scheduler, which aims to apply Ant Colony Optimization techniques [7] to schedule processes on heterogeneous capacity computers. Experiments were carried out to evaluate the proposed algorithm and compare it with other algorithms found in the literature. The results confirm its applicability in heterogeneous cluster computing environments and suggest its potential as an alternative approach for load balancing.

This paper is divided into the following sections: section 2 briefly introduces the basic concepts of Ant Colony Optimization; section 3 describes the proposed load balancing algorithm based on Ant Colonies, named Ant Scheduler; The simulations performed and the results obtained are presented in section 4; section 5 describes the implementation of the proposed algorithm; Finally, section 6 has the main conclusions of this work.

2 Ant Colony Optimization

In the last years, there was a large growth in the research of computational techniques inspired in nature. This area, named Bio-inspired Computing, has provided biologically motivated solutions for several real world problems. Among the Bio-inspired Computing techniques, Artificial Neural Networks (ANN), Evolutionary Algorithms (EA), Artificial Immune Systems (AIS) and Ant Colony Optimization (ACO) can be mentioned.

One of the most promising of these techniques is ACO [7], a meta-heuristic technique based on the structure and behavior of ant colonies that has been successfully applied to several optimization problems [8,9].

Apparently simple organisms, ants can deal with complex tasks by acting collectively. This collective behavior is supported by the release of a chemical substance, named pheromone. During their movement, ants deposit pheromone in their followed paths. The presence of pheromone in a path, on its turn, attracts other ants. Thus, pheromone plays a key role in the information exchange between ants, allowing the accomplishment of several important tasks. A classical example is the selection of the shortest path between their nest and a food source.

In order to formally define ACO, assume four ants and two possible paths, P_1 and P_2 , which link a nest N_E to a food source F_S, such that $P_1 > P_2$. Initially, all the ants (A_1, A_2, A_3 and A_4) are in N_E and must choose between the paths P_1 and P_2 to arrive to F_S.

1. In N_E, the ants (A_1, A_2, A_3 and A_4) do not know the localization of the food source (F_S). Thus, they randomly choose between P_1 and P_2, with the same probability. Assume that ants A_1 and A_2 choose P_1, and ants A_3 and A_4 choose P_2.
2. While the ants pass by P_1 and P_2, they leave a certain amount of pheromone on the paths, τ_{C1} and τ_{C2}, respectively.
3. As $P_2 < P_1$, A_3 and A_4 arrive to F_S before A_1 and A_2. In this moment, $\tau_{C2} = 2$. Since A_1 and A_2 have still not arrived to F_S, $\tau_{C1} = 0$. In order to come back to N_E, A_3 and A_4 must choose again between P_1 and P_2. As in F_S, $\tau_{C2} > \tau_{C1}$, the probability of these ants choosing P_2 is higher. Assume that A_3 and A_4 choose P_2.
4. When A_3 and A_4 arrive to N_E again, τ_{C2} arrives to 4. This increase in τ_{C2} consolidates the rank of P_2 as the shortest path. When A_1 and A_2 arrive to F_S, $\tau_{C2} = 4$ and $\tau_{C1} = 2$. Thus, the probability of A_1 and A_2 coming back to N_E through P_2 becomes higher.

In the previous example, at the beginning, when there is no pheromone, an ant looking for food randomly chooses between P_1 and P_2 with a probability of 0.5 (50% of possibility for each path). When there is pheromone on at least one of the paths, the probability of selecting a given path is proportional to the amount of pheromone on it. Thus, paths with a higher concentration of pheromone have a higher chance of being selected.

It must be observed that most ACO approaches, in spite of being inspired by the problem solving paradigms found in biological ants, do not build replicas of them. Features of real ants may be absent and other additional techniques may be used to complement the use of pheromone.

One of the problems that may arise with the use of pheromone is the stagnation. Suppose, for example, that ants get addicted to a particular path. Sometimes in the future, that path may become congested, becoming nonoptimal. Another problem arises when a favorite path is obstructed and can no longer be used by the ants. In order to reduce this problem, the following approaches have been employed:

Evaporation: Reduce the pheromone values τ_i by a ρ factor to prevent high pheromone concentration in optimal paths, which avoids the exploration of other (new or better) alternatives.

Heuristic: This approach combines pheromone concentration τ_i and a heuristic function η_i. The relative importance of each information is defined by two parameters, α and β.

3 Ant Scheduler

The problem of process allocation in heterogeneous multi-computing environments can be modeled by using graphs. In this case, each process request for execution has the nodes S and T as origin and destination, respectively. S and T are connected by N different paths, each corresponding to a computer in a cluster. This graph is employed to improve the general performance of the system by minimizing the mean congestion of the paths.

The good results obtained by ACO in graph-based problems favor the use of ACO for the optimization of process allocation on heterogeneous cluster computing environments. For such, each initiated process can be seen as an ant looking for the best path starting in the nest in order to arrive as fast as possible to the food source. In this search, each computer can be seen as a path and the conclusion of the program execution as the food source.

The Ant Scheduler algorithm is based on the *ant-cycle* proposed by Dorigo *et al.* [7]. When the computer responsible for the distribution of processes (master) in the cluster is started, each edge in the graph has its pheromone intensity initiated with a value $\tau_i = c$. When a process is launched, it is seen as an ant able to migrate. Thus, this process must select one of the paths (the computers of the cluster) to its destination (complete execution). The probability of an ant choosing a path i is given by equation 1, where τ_i is the pheromone level on path i, η_i is a value associated to the computer i by a heuristic function, and the parameters α and β control the relevance of τ_i and η_i:

$$p_i = \frac{\tau_i^{\alpha} \cdot \eta_i^{\beta}}{\sum_{j=1}^{N} \tau_j^{\alpha} \cdot \eta_j^{\beta}}, \tag{1}$$

$$\Delta_i = \Delta_i + \frac{Q}{T}, \tag{2}$$

$$\tau_i(t+1) = \rho \cdot \tau_i(t) + \Delta_i. \tag{3}$$

In this paper, this heuristic function is proportional to the load of the ith computer. The denominator is the sum of the pheromone levels weighted by the heuristic function and controlled by the parameters α and β. When an ant arrives to its destination (when a process finishes), it deposits a Δ amount of pheromone in the covered path (equation 2: where Q is a constant and T is the time spent by the ant to arrive at its destination (the process running time)).

In order to prevent an addiction to a particular computer, the paths face continuous pheromone evaporation. Thus, in regular time intervals, the amount of pheromone changes according to the rule of equation 3, where ρ is a coefficient such that $(1 - \rho)$ represents the pheromone evaporation between t and $t + 1$. Additionally, Δ_i is reseted ($\Delta_i = 0$) in regular time intervals.

One problem with this approach is the possibility of a poor performance due to the different range of values for τ_i and η_i. In order to overcome this problem, these values are normalized using a logarithmic scale, modifying the equation 1 and originating the equation 4:

$$p_i = \frac{(\log \tau_i)^{\alpha} \cdot (\log \eta_i)^{\beta}}{\sum_{j=1}^{N} (\log \tau_j)^{\alpha} \cdot (\log \eta_j)^{\beta}}. \tag{4}$$

Another problem found was the frequent allocation of a low value, between 0 and 1, to τ_i, making $\log \tau_i < 0$, leading to unrealistic values for the probability function. This problem was solved by using $\log \epsilon + \tau_i$ instead of $\log \tau_i$, where $\epsilon = 1$. This resulted to the equation 5:

$$p_i = \frac{(\log \epsilon + \tau_i)^\alpha \cdot (\log \epsilon + \eta_i)^\beta}{\sum_{j=1}^{N} (\log \epsilon + \tau_j)^\alpha \cdot (\epsilon + \log \eta_j)^\beta}. \quad (5)$$

Algorithm 1. Ant Scheduler: process started

Choose a computer with probability p_i, calculated using equation 5
Schedule process on chosen computer

Algorithm 2. Ant Scheduler: process finished

Update the amount of pheromone $\varDelta_i$ using equation 2

Algorithm 3. Ant Scheduler: pheromone evaporation

loop
  for all i such that i is a cluster node **do**
    Update the amount of pheromone τ_i using equation 3
    Reset the amount of pheromone $\varDelta_i$ ($\varDelta_i = 0$)
  end for
end loop

The Ant Scheduler is composed of the Algorithms 1, 2 and 3. The first algorithm is executed when a new process, with possibility of migration, is initiated. When a process completes its execution, the second algorithm starts. The third algorithm is periodically executed, in order to perform the pheromone evaporation.

4 Simulation Results

Several experiments have been carried out on environments with 32 computers for the evaluation of the Ant Scheduler algorithm behavior. The Ant Scheduler parameters used were $\alpha = 1$, $\beta = 1$, $\rho = 0.8$ and $Q = 0.1$. Parallel applications of up to 8, 64 and 128 tasks have been evaluated. This configuration allows the evaluation of the algorithm in situations where there are many tasks synchronized with others, that is, tasks that communicate among themselves to solve the same computing problem.

The workload imposed by such applications follows the workload model by Feitelson[1][10]. This model is based on the analysis of six execution traces of the following production environments: 128-node iPSC/860 at NASA Ames; 128-node IBM SP1 at

[1] http://www.cs.huji.ac.il/labs/parallel/workload/models.html

Argonne; 400-node Paragon at SDSC; 126-node Butterfly at LLNL; 512-node IBM SP2 at CTC; 96-node Paragon at ETH.

According to this model, the arrival of processes is derived from an exponential probability distribution function (`pdf`) with mean equal to $1,500$ seconds. This model was adopted to simulate and allow a comparative evaluation of Ant Scheduler and other algorithms found in the literature.

In order to carry out the experiments and evaluate the scheduling algorithm proposed in this study, the authors used the model for creation of heterogeneous distributed environments and evaluation of the parallel applications response time - UniMPP (*Unified Modeling for Predicting Performance*) [11]. The adopted model is able to generate the mean execution time of the processes submitted to the system. The mean response time is generated after reaching the confidence interval of 95%.

In this model, every processing element (PE), PEM_i, is composed of the sextuple $\{pc_i, mm_i, vm_i, dr_i, dw_i, lo_i\}$, where pc_i is the total computing capacity of each computer measured in instructions per time unit, mm_i is the main memory total capacity, vm_i is the virtual memory total capacity, dr_i is the hard disk reading throughput, dw_i is the hard disk writing throughput, and lo_i is the time between sending and receiving a message.

In this model, every process is represented by the sextuple $\{mp_j, sm_j, pdfdm_j, pdfdr_j, pdfdw_j, pdfnet_j\}$, where mp_j represents the processing consumption, sm_j is the amount of static memory allocated by the process, $pdfdm_j$ is the probability distribution for the memory dynamic occupation, $pdfdr_j$ is the probability distribution for file reading, $pdfdw_j$ is the probability distribution for file writing, and $pdfnet_j$ is the probability distribution for messages sending and receiving.

In order to evaluate the Ant Scheduler algorithm, a class was included in the object-oriented simulator [2] [11]. This class implements the functionalities of Ant Scheduler and has been aggregated to the UniMPP model simulator to generate the mean response times of an application execution. These results were used to evaluate the performance of Ant Scheduler and to allow comparisons with other algorithms.

4.1 Environment Parameterizations

Experiments were conduced in environments composed of 32 computers. In these experiments, each PEM_i for the UniMPP model was probabilistically defined. The parameters pc_i, mm_i, vm_i, dr_i, dw_i were set by using an uniform probability distribution function with the mean of $1,500$ Mips (millions of instructions per second), $1,024$ MBytes (main memory), $1,024$ MBytes (virtual memory), 40 MBytes (file reading transference rate from hard disk) and 30 MBytes (file writing transference rate to hard disk). These measures were based on the actual values obtained using a group of machines from our research laboratory (Distributed Systems and Concurrent Programming Laboratory). These measures followed the benchmark proposed by Mello and Senger [11][3]. These parameter values and the use of probability distributions allow the creation of heterogeneous environments to evaluate the Ant Scheduler algorithm.

[2] SchedSim - available at website http://www.icmc.usp.br/˜mello/outr.html

[3] Available at http://www.icmc.usp.br/˜mello/

The Feitelson's workload model was used to define the occupation parameter (in Mips) of the processes (or tasks) that take part of the parallel application. The remaining parameters required for the UniMPP to represent a process were defined as: sm_j, the amount of static memory used by the process, based on an exponential distribution with a mean of 300 KBytes; $pdfdm_j$, the amount of memory dynamically allocated, defined by an exponential distribution with a mean of $1,000$ KBytes; $pdfdr_j$, the file reading probability, defined by an exponential distribution with a mean of one read at each $1,000$ clock ticks, same value used to parameterize the writing in files ($pdfdw_j$); $pdfnet_j$, the receiving and sending of network messages, parameterized by an exponential distribution with a mean of one message at each $1,000$ clock ticks.

During the experiments, all computers were located at the same network, as a ordinary cluster. Within the network, the computers present a delay (RTT - Round-Trip Time according to the model by Hockney [12]) of 0.0001 (mean value extracted by the *net* benchmark by Mello and Senger [11] for a Gigabit Ethernet network).

4.2 Algorithms Simulated

The performance of Ant Scheduler is compared with 5 other scheduling and load balancing algorithms proposed in literature: `DPWP` [13], `Random`, `Central`, `Lowest` [3], `TLBA` [5] and `GAS` [6].

The `DPWP` (*Dynamic Policy Without Preemption*) algorithm performs the parallel applications scheduling taking into account a distributed and heterogeneous execution scenario [13]. This algorithm allows the scheduling of the applications developed on PVM, MPI and CORBA. The details involved in the task attributions are supervised by the scheduling software, AMIGO [14][4].

The load index used in this algorithm is the queue size of each PE (processing element). Through this index, the load of a PE is based on the ratio between its number of tasks being executed and its processing capacity. The processing capacity is measured by specific *benchmarks* [14,15]. The `DPWP` scheduling algorithm uses load indexes to create an ordered list of PEs. The parallel application tasks are attributed to the PEs of this list, in a circular structure.

The `Lowest`, `Central` and `Random` algorithms were investigated for load balancing in [3]. These algorithms are defined by two main components: LIM (*Load information manager*) and LBM (*Load balance manager*). The first component is responsible for the information policy and for monitoring the computers' load in order to calculate the load indexes. The latter defines how to use the collected information to find out the most appropriate computer to schedule processes. The approach followed by these components to perform their tasks allows the definition of distinct algorithms. These algorithms differ from the scheduling algorithms by being designed to perform the load balance, thus there is no global scheduling software to which the applications are submitted. In fact, each PE should locally manage the application tasks that reach it, initiating them locally or defining how another PE will be selected to execute tasks.

[4] We have compared our results to this work, because it was also developed in our Laboratory.

The Lowest algorithm aims to achieve the load balance by minimizing the number of messages exchanged among its components. When a task is submitted to the environment, the LIM receiving the request defines a limited set of remote LIMs. The loads of the PEs of this set are received and the idlest PE is selected to receive the task.

The Central algorithm employs a master LBM and a master LIM. Both of them centralize the decision making related to the load balance. The master LIM receives the load indexes sent by the slave LIMs. The master LBM receives the requests to allocate processes to the system and uses the information provided by the master LIM to make these allocations.

The Random algorithm does not use information regarding the system load to make decisions. When a task is submitted to the execution environment, the algorithm randomly selects an PE. The load index used by the Lowest and Central algorithms is calculated based on the number of processes in the execution queue. Zhou and Ferrari [3] have observed that the Lowest and Central algorithms present similar performance and that the Random algorithms present the worst results of all. They also suggested the Lowest algorithm for distributed scenarios, because it is not centralized.

The TLBA (*Tree Load Balancing Algorithm*) algorithm aims at balancing loads in scalable heterogeneous distributed systems [5]. This algorithm creates a logical interconnection topology with all PEs, in a tree format, and performs the migration of tasks in order to improve the system load balance.

The GAS (*Genetic Algorithm for Scheduling*) algorithm uses Genetic Algorithms to propose optimized scheduling solutions [6]. The algorithm considers knowledge about the execution time and applications' behavior to define the most adequate set of computing resources to support a parallel application on a distributed environment composed of heterogeneous capacity computers. GAS uses the crossover and mutation operators to optimize the probabilistic search for the best solution for a problem. Based on Genetics and Evolution, Genetic Algorithms are very suitable for global search and can be efficiently implemented in parallel machines.

4.3 Comparison with Other Algorithms

For the validation of the Ant Scheduler algorithm, its performance was compared with results obtained by the five algorithms previously described. For such, the authors carried out simulations where all these algorithms were evaluated running parallel applications composed of different number of tasks. Figures 1.a and 1.b show the process mean response times for parallel applications with up to 64 and 128 tasks, respectively.

Random had the worst results, while Ant Scheduler presented the best performance. The poor performance obtained by GAS can be explained by the fact that its execution time increases according to the number of computers. This occurs due to the use of larger chromosomes (this approach is based on Genetic Algorithms), which have to be evaluated by the fitness function. This evaluation requires a long time, which is added to the scheduling cost, jeopardizing the process execution time. It is hard to observe the curve for Ant Scheduler in Figure 1.a due to the small mean response times in comparison with the other algorithms.

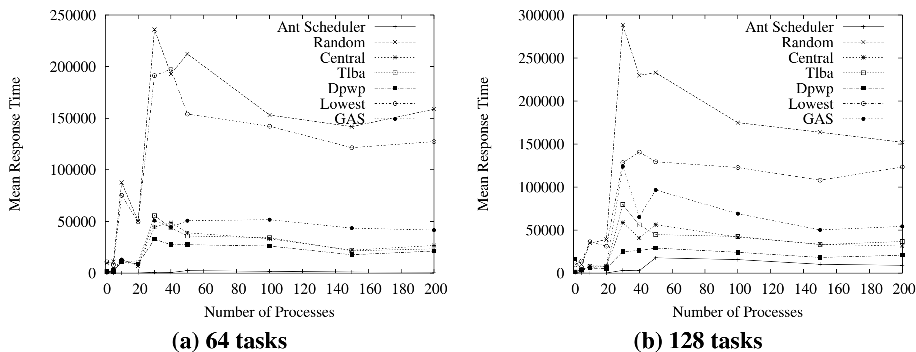

Fig. 1. Simulation results

These results show that, in all the scenarios investigated, the Ant Scheduler presented the best performance.

5 Implementation

In order to allow real experiments, Ant Scheduler was implemented using the Linux kernel 2.4.24. This implementations uses the process migration service of the openMosix[5] patch. openMosix is a software designed to balance the load of clusters by distributing processes.

The implementation was performed by adding a set of traps inside the Linux kernel. The first trap was implemented in the system call *do_fork*. Whenever a new process is started, *do_fork* is called. This system call executes the first trap of Ant Scheduler, which chooses the node where the new process will run. This phase is based on the pheromone level and the computing capacity of each node. Similarly, when a process finishes, the system call *do_exit* is made. This system call executes the second trap of Ant Scheduler, which updates the amount of pheromone in the computer (ant's path) where the process was running.

These traps were implemented in a kernel module by using function pointers, allowing simple changes to use another process scheduling algorithm. When the module is loaded, it registers its functions (traps). This module also starts a thread that periodically updates the pheromone level of each computer applying the equation 3.

Experiments were carried out to evaluate the process execution time for an environment using Ant Scheduler and openMosix on a set of five Dual Xeon 2.4 Ghz computers. Table 1 presents the results in process mean execution time (in seconds) for a load of 10 low-load, 10 mean-load and 10 high-load applications executing simultaneously. According to these results, the use of Ant Scheduler reduced the mean response time.

[5] Openmosix is a Linux kernel patch developed by Moshe Bar which allows automatic process migration in a cluster environment – Available at http://openmosix.sourceforge.net/

Table 1. Experimental Results

Experiment	without Ant Scheduler	with Ant Scheduler
1	351.00	327.00
2	351.00	336.00
3	351.00	318.00
4	354.00	321.00
5	351.00	318.00
6	351.00	333.00
7	351.00	321.00
8	351.00	336.00
9	348.00	309.00
10	348.00	315.00
Mean	350.70	323.40
Std Dev	1.615	8.777

In order to evaluate the statistical significance of the results obtained, the authors applied the Student's t-test. In this analysis, the authors used the standard error s_x for small data samples [16], given by equation $s_x = \frac{s}{\sqrt{n}}$.s is the standard deviation and n is the number of samples. Applying the equation, the standard errors of 0.51 and 2.775 were obtained without Ant Scheduler and with Ant Scheduler, respectively.

In the test, the authors propose the null hypothesis (from hypothesis test) $H_0 : \mu_{with} = \mu_{without}$, with the alternative hypothesis $H_A : \mu_{with} < \mu_{without}$ to evaluate whether the results are statistically equivalent. The hypothesis H_0 considers the results of the Ant Scheduler and the standard openMosix to be similar. If the test is rejected, the alternative hypothesis H_A is accepted. This hypothesis considers the process mean response time for the environment adopted. The processes are distributed using the Ant Scheduler is lower, what confirms the superiority of Ant Scheduler.

The significance level used for one-tailed test is $\alpha = 0.0005$. μ_{with} is the process mean response time with Ant Scheduler; $\mu_{without}$ is the process mean response time with the standard openMosix. For the adopted significance level α, the data sets have to present a difference of at least 4.781 in the t-test to reject the hypothesis. This value is found in tables of critical values for the *t*-student distribution.

Applying the equation $t = \frac{\mu_{without} - \mu_{with}}{s_x}$, the value 9.83 is found, confirming that the results present statistic differences with $p < 0.005$, rejecting the hypothesis H_0. In this way the hypothesis H_A is valid and the system with Ant Scheduler presents better results than standard openMosix.

By applying statistic tools[6] over the data sets, it is possible to find the most precise $\alpha = 0.0000018$ for a one-tailed test. This value shows how many times the alternative hypothesis is true. In this case, H_A can be considered true in $9,999,982$ of $10,000,000$ executions, showing that Ant Scheduler reduces the response time. Only in 18 of these

[6] The authors applied the Gnumeric tool in this analysis – a free software tool licensed under GNU/GPL terms.

executions the results of Ant Scheduler and openMosix did not present significant statistical differences.

6 Conclusions

In this work, the authors proposed a new approach for load balance in heterogeneous computers using an algorithm based on Ant Colony Optimization. The proposed approach was motivated by the successes obtained by this technique in several real world problems.

This algorithm, named Ant Scheduler, had its performance compared with five other algorithms found in the literature. The simulation results, where the proposed algorithm presented a performance superior to the other algorithms investigated, suggest the potential of the Ant Scheduler algorithm for heterogeneous cluster computing environments. These results have motivated its implementation to validate the theoretical concepts. The algorithm was implemented in a real Linux environment. This implementation was evaluated through experiments and compared with the standard scheduling in openMosix. In these experiments, Ant Scheduler also has presented good results.

Acknowledgement

The authors thank to the Fapesp Brazilian Foundation (process number 2004/02411-9).

References

1. Shivaratri, N.G., Krueger, P., Singhal, M.: Load distributing for locally distributed systems. IEEE Computer **25**(12) (1992) 33–44
2. Krueger, P., Livny, M.: The diverse objectives of distributed scheduling policies. In: Seventh Int. Conf. Distributed Computing Systems, Los Alamitos, California, IEEE CS Press (1987) 242–249
3. Zhou, S., Ferrari, D.: An experimental study of load balancing performance. Technical Report UCB/CSD 87/336, PROGRES Report N.o 86.8, Computer Science Division (EECS), Universidade da California, Berkeley, California 94720 (1987)
4. Theimer, M.M., Lantz, K.A.: Finding idle machines in a workstation-based distributed system. IEEE Transactions on Software Engineering **15**(11) (1989) 1444–1458
5. Mello, R.F., Trevelin, L.C., Paiva, M.S., Yang, L.T.: Comparative analysis of the prototype and the simulator of a new load balancing algorithm for heterogeneous computing environments. In: International Journal of High Performance Computing and Networking. Volume 1, No.1/2/3. Interscience (2004) 64–74
6. Senger, L.J., de Mello, R.F., Santana, M.J., Santana, R.H.C., Yang, L.T.: Improving scheduling decisions by using knowledge about parallel applications resource usage. In: Proceedings of the 2005 International Conference on High Performance Computing and Communications (HPCC-05), Sorrento, Naples, Italy (2005) 487–498
7. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B **26** (1996) 1,13

8. Shmygelska, A., Hoos, H.H.: An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. BMC Bioinformatics (2005) 1–22
9. Acan, A.: An external memory implementation in ant colony optimization. In: Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence, Brussels, Belgium (2004) 73–82
10. Feitelson, D.G., Jette, M.A.: Improved utilization and responsiveness with gang scheduling. In Feitelson, D.G., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Springer (1997) 238–261 Lect. Notes Comput. Sci. vol. 1291.
11. de Mello, R.F., Senger, L.J.: Model for simulation of heterogeneous high-performance computing environments. In: 7th International Conference on High Performance Computing in Computational Sciences – VECPAR 2006, Springer-Verlag (2006) 11
12. Hockney, R.W.: The Science of Computer Benchmarking. Soc for Industrial & Applied Math (1996)
13. Araújo, A.P.F., Santana, M.J., Santana, R.H.C., Souza, P.S.L.: DPWP: A new load balancing algorithm. In: ISAS'99, Orlando, U.S.A. (1999)
14. Souza, P.S.L.: AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos. PhD thesis, IFSC-USP (2000)
15. Santos, R.R.: Escalonamento de aplicações paralelas: Interface amigo-corba. Master's thesis, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (2001)
16. W.C.Shefler: Statistics: Concepts and Applications. The Benjamin/Cummings (1988)