BOUNDED ANT COLONY ALGORITHM FORTASK ALLOCATION ON ANETWORK OF HOMOGENEOUS PROCESSORSUSING A PRIMARY SITE (BTS-ACO)

Buthayna Al-Sharaa[1] and Tamara Al-Qublan[2]

[1]Departement of Electrical Engineering, AL Balqa Applied University, Al Huson, Jordan
buthayna74@hotmail.com
[2]Departement of Information Technology, AL Balqa Applied University, Al Huson, Jordan
tqablan@hotmail.com

ABSTRACT

Efficient scheduling of tasks for an application is considered a crucial aspect in distributed systems to achieve a superior performance. This paper presents a task scheduling algorithm base on the Ant Colony Optimization (BTS-ACO). This algorithmdepends on an initial bound on each processor to control the procedure of task allocation. Herein, the priority of tasks is to processor with the minimal load. The algorithm investigates the effect of scheduling sorted (SLoT) and random (RLoT) list of tasks. The performance of the algorithm is demonstrated by the time taken for producing effective schedules, makespan of the schedule, and load balance of the models. The results show that BTS-ACO solution with sorted list has better performance than random list.

KEYWORDS

Ant colony optimization Task scheduling, parallel programming, load balancing, Makespan

1. INTRODUCTION

The problem of parallel processing in distributed and computers cluster systems is continuing to gain big interest. A parallel program is a program that can be divided into a collection of subtasks. These subtasks are usually called tasks or processes [1]. Some of tasks depend on the completion of other tasks; others can be executed at the same time, which increases parallelism of the problem. Tasks can be executed in sequence or at the same time on one or more processors.
A computer cluster consists of multiple computers that are linked through a LAN. The networked computers essentially act as a single, much more powerful machine. A computer cluster offers faster processing speed, larger storage capacity, better data integrity and the ability to handle large computational load [2]. When a cluster is built, usually one computer is chosen as a root or primary node. The responsibility of the root is to receive a task, divide it into subtasks, then delivers these subtasks to the cluster members. The root continuous to communicate with the nodes in the cluster. When the nodes finish executing their share, they send the result back to the root which will integrate them into its final form.

Task scheduling is the problem of assigning tasks to the processors in the system in a manner that will minimize the makespan, while assuring the correctness of the result. Makespan is the completion time of the last task relative to the start time of the first task. The scheduling problem is a NP-hard problem[3] and it is not trivial. Many task scheduling algorithms have been

DOI : 10.5121/ijcsit.2013.5313

presented. The objectives of these algorithms are usually to balancethe load among processors, reduce the execution time of the parallel program, minimize the makespan and others.
Task scheduling algorithms can be static or dynamic approaches. Static or off-line scheduling algorithms assume that all information about the scheduling problem is known in advance, and the solution to the problem is computed and found before the system starts running. On the other hand, dynamic or on-line scheduling algorithms compute the solution for tasks while the scheduling process is running. Dynamic scheduling is flexible and can cope with any unexpected changes in the system during the scheduling process.

Many methods of scheduling techniques exist. These methods can be classified into three categories; graph theory based methods [4], mathematical models based methods [5], and heuristic techniques [6][7][8][9]. Several numbers of heuristics have been used and the most well-known of them are; the iterative improvement algorithms, the probabilistic optimization algorithms, and the constructive heuristics.

Ant colony algorithms are meta-heuristic algorithms inspired by the behavior of real ant colonies. Real ants find food sources by indirectly communicating together by means of releasing certain amount of pheromones while walking.

Ants follow paths with higher amount of pheromones. These paths are usually the optimal solutions. The ant colony algorithm is applied to solve many difficult optimization problems in different applications [10][11][12][13][14][15][16]. In this paper we apply a modified version of ant colony algorithm to task scheduling in a computer cluster system. Simulation of the proposed algorithm is presented. The simulator defines the different parameters of the system such as execution time of tasks, the criteria of a scheduler, etc. Finally, the paper compares the performance of various task scheduling approaches.

The rest of the paper is organized as follows. In section 2, the related work is presented. Section 3 discusses the problem statement. Section 4 describes the methodology and design, and briefly illustrates how to implement a task scheduling system using an Ant Colony Optimization (ACO) algorithm. Section 5 presents the experimental results and the discussions. Section 6 concludes the paper and proposes future work for the problem.

2. RELATED WORK

Task scheduling is an NP-problem. Many task scheduling algorithms have been presented. These methods can be classified into three categories; graph theory methods, mathematical methods, and heuristic techniques. Some of the heuristic algorithms are min-min [17], the fast greedy [17] and Ant colony algorithms [18]. In min-min method, the task with minimum completion timeis selected first and assigned to a processor. This algorithm has fast scheduling time, but poor load balancing [19][20]. Fast Greedy assigns each task, in arbitrary order, to the processor with the minimum completion time [19]. Some algorithms force some conditions to improve the performance such as in [21], where the task can be moved from one machine to another. This action improves the load balancing, but on the other hand the traffic in the system is increased.

3. PROBLEM STATEMENT

The system consists of a clusterof m processing machines connected via a network. Let
P = {P1, P2… Pk} denote the set of processors in the system.All processors in the cluster are fully connected.The processors in the cluster are homogeneous, meaning that the execution time

of a specific task is the same on any processor. One of the processors in the cluster is chosen as a root. The function of the root is to divide the original task into independent subtasks. After these subtasks are delivered to the processors in the cluster and executed, the results are sent back to the root which integrates them into their final format [1].

The Tasks Set T = {T1, T2…Tn) has n tasks. The set T stores the execution time of the tasks to be schedules. The number of processors in the system is less than the number of tasks. So each processor will probably have more than one task. The number of tasks assigned to a processor will depend on the execution time of thetasks and on the load already assigned to that CPU.
Each processor should wait for some time before it can execute its subtask. This is the time needed to transmit the task over the communication channel. Usually the execution time of the subtask is much higher than its transmission time, so we will neglect the transmission time in our calculations.

All processors involved in the computing process of the parallel tasks must stop at the same time to obtain the minimum finish time. This optimality criterion has beenrigorously proved in the literature [22]. The goal is to minimize the maximum total processing time, also known as the makespan, on any processor. So if we have K processors in the system, then ET1= ET2=…. = E Tk, where(E Ti) is the total execution time of processor i [2].

An upper bound value is used to control the scheduling of tasks over the processors. This value is unique for the system. Each time a task is delivered to a processor, the total execution times of all tasks (ETi) assigned toprocessor I (including the new task) is tested against the bound. If ETi exceeds the bound, then the task will be sent to another processor. The value of the bound is calculated using equation 1 below. m,in the equation, is the number of tasks to be scheduled, and k is the number of processors in the cluster.

$$Bound = \frac{\sum_{j=0}^{j=m} T_j}{k} \qquad \text{Equation 1}$$

The bound can be increased during the scheduling processor. This action is done in case there is a task that can't be delivered to any processor because the load of each processor plus the new task exceeds the bound.

At the end of the scheduling process the finish time of each processor is examined. The ant colony algorithm is used to departure tasks to CPUs in the system. Our goal is to make an equal finish time for all processors. Unfortunately, this will be difficult since each task have different execution time.

As mentioned above, this paper proposed different approaches in which tasks were stored and picked from the array in order to be delivered to the processors.

Case A: Random List of Tasks (RLoT)

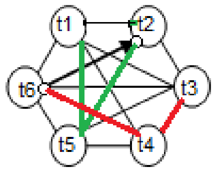
With this policy, a task is picked randomly from the array and sent to a processor.This policy is the simplest to implement since it involves only a negligible amount ofOverhead when generating the tasks.

Case B: Sorted List of Tasks (SLoT)

In this case the tasks in the array T are sorted in descending order, according to their execution time, and picked in sequence. Unfortunately, extra computational time will be needed for sorting the array. Each time a task is picked and delivered to a different processor. Since we have a sorted task list, the first k tasks with higher execution times will be picked first, and then the next k tasks with lower execution times will be picked, and so on. Using this method we assign the processor's tasks with high and small execution times in an attempt to balance the load given to these processors. The ant colony algorithm will benefit from this technique in setting initial pheromone values between small and big ones. This will help in building schedules faster and achieving a higher degree of load balancing.

4. ANT COLONY OPTIMIZATION FOR TASKS SCHEDULING

Task scheduling problem is represented as a complete graph G = (V, E) where node (V) represents a task and the edge (E) represents a path between two tasks. Each edge in the graph is assigned a pheromone trial t. An illustrative example is depicted in Figure 1.



Full Task Set = {t1,t2,t3,t4,t5,t6}, Two processors.
Allocate {t1,t2,t5} to processor 1,And allocate {t3,t4,t6} to processor 2, where number of processor equal 2

Figure 1: An illustration for constructing a solution

The proposed algorithm for tasks scheduling (BTS_ACO) follows the standard ACO algorithm that is used for static combinatorial optimization problems but with new features. The algorithm performs as follow: at each cycle, every ant constructs a solution and then the pheromone trials for each ant are updated with a value which depends on the solution constructed by the ants. The ant that constructs the best solution will have the higher amount of pheromone update. The algorithm stops iterating when the maximum number of cycles is reached. The Pseudo-code of the complete algorithm (BTS_ACO) is presented in Figure 2.

```
Algorithm:  (BTS_ACO)

Initializing data
For each ant k in 1 … num_ant  do
{
    produce candidate solution randomly
    Update pheromone
    Save best solution
    Update pheromone for best solution
 }
Repeat
 {
    For each ant k in 1 … num_ant  do
```

```
        {  // begin for ant
             Construct solution as the following
           -  Select the first task t randomly and assign it to a processor
        T=T-t
             Repeat
               {
                      Select tasks depending  on the property value p as given in
          equation 2
               }
               WhileT! =θ
            Update pheromone
            Save best solution
            Update pheromone for best solution
        }  //end for ant
         Update pheromone trial as follows:
               if (Ms<Msg) then
               $Bs_{star}$=$S_{save}$
            Msg=Ms
                For every edge between two tasks  BS$_{star}$ do
                  $t$(t1,t2)=$t$(t1,t2)+q
               For every edge between two tasks  do
                  $t$(t1,t2)=  ρ* $t$(t1,t2)
    } Until (terminated condition is reached)
```

Figure 2: The steps of the BTS_ACO algorithm.

Pheromone Trials and Heuristic Information

In this step the pheromone matrix is initialized to equal amount of pheromone. In the next step, every ant constructs its own solution; after that the pheromone trial for each edge in the solution is updated with a value which depends on the solution constructed by the ants. Typically, the edges of the best solution will have the higher amount of pheromone update.

Constructing a Solution

In BTS_ACO algorithm each ant starts from a randomly selected task, and then it selects the next task from those unselected tasks depending on the property value p as given in the next formula [23].

$$p_{a,c}^{k}(t) = \frac{t_{a,c}^{\alpha}(t) * h_{a,c}^{\beta}(t)}{\sum_{i \in u} t_{a,c}^{\alpha}(t) * h_{a,c}^{\beta}(t)} \qquad \text{Equation 2}$$

Where k denotes the ant number,

$p_{a,c}^{k}$ is the probability with which ant k chooses to select task a with set of tasks in processor c,
t denotes the iteration numbers,

u denotes the set of task that are not visited yet,

$t_{a,c}$ is the sum of pheromone value between task a and set of task in processor c,

$h_{a,c}$ is the heuristic function which was chosen to be the inverse of the maximum difference of execution time between set of processor if we add task a to processor c at iteration t,
α and β are parameters that control the pheromone trials and the heuristic information.
The construction process is stopped when the maximum number of cycles is achieved.

Pheromone Update

After each ant has constructed a solution, pheromone trails are updated on each edge. The ant that constructs the best solution (*bs*) will have the higher amount of pheromone update. In BTS_ACO, pheromone is updated according to the following formula:

$$t_s(t+1) = \rho t_s(t) + {}^{q}\!/_{\rho} \qquad \text{Equation 3}$$

Where t_s is the amount of pheromone between tasks in schedule s at time t,
t_s (t+1) is the amount of pheromone between tasks in schedule s at next iteration,
q, ρ is a given constant values.

While for other schedule, the pheromone trails are updated using the following formula:

$$t_j(t+1) = \rho t_j(t) \qquad \text{Equation 4}$$

5. Experimental Results

In this section, we used the simulation results to show the performance of the proposed BTS_ACO algorithm in a task scheduling system.The simulation programs are performed on Intel Core i3 2.2 GHz machine with 4 GB RAM. Table1 below shows the computational workloads used in the simulation.

Table1. Taskworkloadparameters

Parameters	**Values**
Number of tasks	8-32 task
Execution time of tasks	10- 800 (seconds)
Number of CPUs	4 homogeneous CPUs

The ACO algorithm parameters values are shown in Table 2.

Table2.Theantparametersvalues

A	**B**	**Lamda**	**Ants**	**q0**
0.5	1	0.9	4	0.1

To show the performance of BTS_ACO for RLoT and SLoT, the number of tasks was varied from 8 to 16 to 32 and measured the load balance, time of schedule creation and makespan. Information about the tasks chosen is shown in table 3.

Table3. Task sets information

Number of tasks	**Total Execution Time(sec)**	**Average Execution time (sec)**
8 task	2165	541
16 task	4565	1141
32 task	8144	2036

Figure 3 shows the comparison of RLoT and SLoT with respect to load balancing. The numbers shown for load balancing is actually the difference between the maximum load and minimum load among all CPUs in the system for a certain schedule. As this number decreases this indicates that the system is capable of distributing the tasks fairly to the CPUs. Load Balance of zero means that all the CPUs have the same amount of work to do. SLoT is capable of balancing the load more effectively to CPUs as shown in the figure.

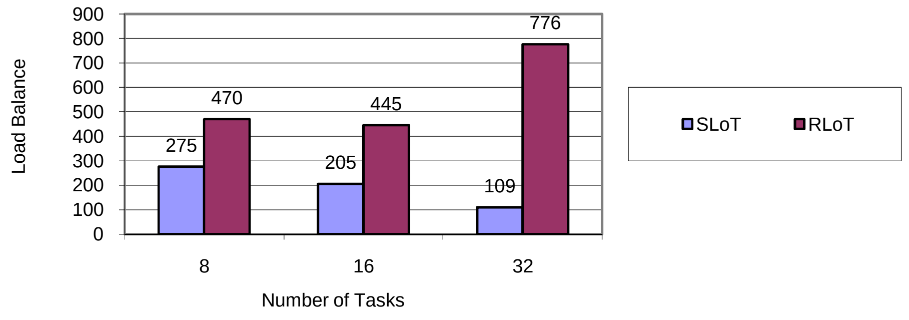


Figure3. Load Balancing capabilities of SLoT and RLoT

Figure 4 shows the time each model needs to create the best schedule. As shown in the figure SLOt outperforms RLoT in this point.

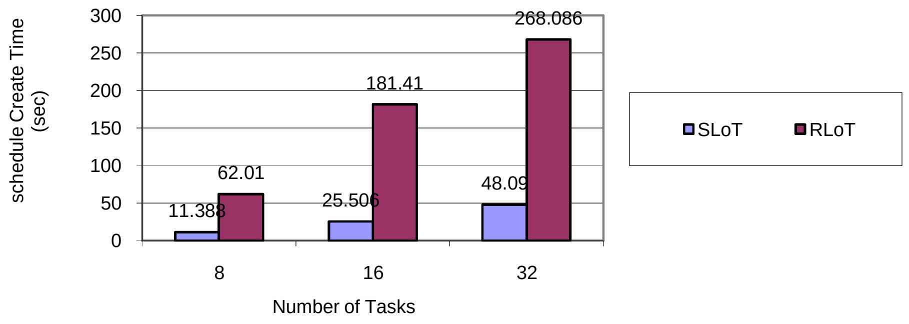


Figure4.Schedule create time of SLoT and RLoT

Figure 5 shows the makespan time of tasks for each model. As mentioned above the makespan is the time needed to complete all the tasks and take results. Again the performance of SLoTis better than that of RLoT.The figure also shows that as the number of tasks increases, the difference between makespan and the average Execution time shown in Table3 becomes smaller. The reason for this is that with higher number of tasks, there will be more tasks and alternatives to schedule.

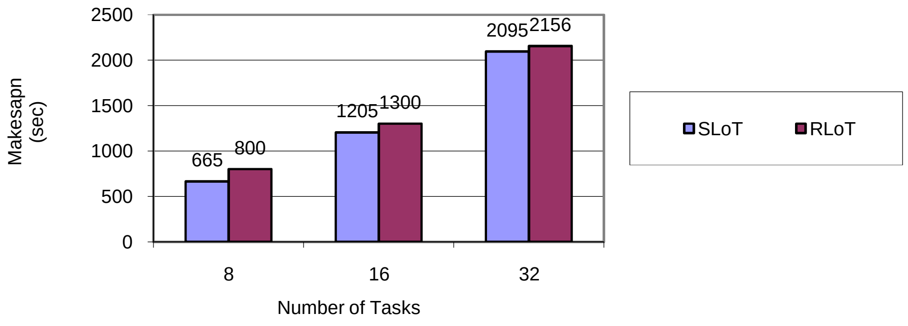


Figure5.Makespan time of SLoT and RLoT

6. CONCLUSION AND FUTURE WORK

In this paper we applied the ant colony optimization to the task allocation problem. The tasks to be scheduled where presented in a random (RLoT) and sorted (SLoT) form. The performanceof BTS-ACO using SLoT outperforms theperformance of BTS-ACO using RLoT in terms of load balancing, schedule create time and makespan time. The future work will examine the performanceof BTS-ACO in a fault tolerant system in addition to applying other different heuristic based algorithms for task allocation problem.

REFERENCES

[1] T Vetri Selvan, Mrs P Chitra, Dr P Venkatesh,(2009) "Parallel Implementation of Task Scheduling using Ant Colony Optimization" , International Journal of Recent Trends in Engineering, Vol. 1, No. 1,pp339-343.

[2] Sameer Bataineh, Buthayna Al-Sharaa and Naheel Qudan, (2004) "Task Allocation on Fault-Tolerant Network of ProcessorsUsing a Primary Site Approach", WSEAS TRANSACTIONSon COMPUTERS, Vol 3,no 4,pp909-916.

[3] Dr Apurva Shah And Vinay Hansora, (2011) "A Modified Genetic Algorithm For Process Scheduling In Distributed System" IJCA Special Issue On "Artificial Intelligence Techniques - Novel Approaches & Practical Applications"AIT.

[4] C.C.Shen, & W.H.Tsai, (1985)"A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Using a Minimax Criterion", IEEE Trans.On Computers, vol. 34, no. 3, pp197-203.

[5] P.Y.R.Ma, E.Y. S. Lee and J.Tsuchiya, (1982) "A Task Allocation Model for Distributed Computing Systems", IEEE Trans. On Computers, vol. 31, no. 1, pp. 41-47.

[6] G. L. Park(2004) "Performance Evaluation of a List Scheduling Algorithm In Distributed Memory Multiprocessor Systems", International Journal ofFuture Generation Computer Systems,vol. 20,pp. 249-256.

[7] C.I.Park and T.Y.Choe, (2002)"An optimal scheduling algorithm based on task duplication", IEEE Trans. on Computers, vol. 51,no. 4,p. 444–448.
[8] C. M. Woodside, and G. G. Monforton, (1993)"Fast Allocation of Processes in Distributed and Parallel Systems", IEEE Trans. On Parallel and Distributed Systems, 4(2), pp. 164-174.
[9] A. K. Sarje and G. Sagar, (1991)"Heuristic Model for Task Allocation in Distributed Computer Systems", Proc. of the IEEE, vol, 138, no.5, pp 313-318.
[10] Colorni, A., Dorigo, M., and Maniezzo, V., and Trubian, M., (1994) "Ant System for Job-Shop Scheduling," Belgian Hournal of Operations Research, Statics and Computer Science (JORBELL), Vol, 34, pp39-53.
[11] Colorni, A., Dorigo, M., and Maniezzo, V.(1992) "Distributed Optimization by ant Colonies, " Proceedings of the first European Conference on Artificial Life, Paris, France, F. Varela and P. Bourgine (Eds), Elsevier Publishing,.pp 134-142.
[12] Dorigo, M., and Gambardella, L.M., (1997)"Ant Colony System:A Cooperative Learning Approach to the Traveling Salesman Problem, "IEEE Trasactions on Evolutionary Compution, Vol. 1, No., 1, pp.53-66.
[13] Dorigo, M., Maniezzo, V., and Colorni, A., (1996) "Ant System: Optimization by a Colony of Cooperating Agents, " IEEE Transactions on Systems, Man, and Cybernnetics-Part B, Vol. 26, No. 1, pp. 29-41.
[14] Maniezzo, V. and Carbonaro, A., (2000) "an Ants Heuristic for the Frequency Assignment Problem," Future Generation Computer Systems, Vol. 16, pp. 927-935.
[15] Maniezzo, V., Colorni, A., and Dorigo, M., (1994) "The Ant System applied to the Quadratic Assignment Problem", IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 5, pp769-778.
[16] Merkle, D., Middendorf, M., and Schmeck, H.(2002)"Ant Colony Optimization for Resource-Constrained Project Scheduling", IEEE Transactions on Evolutionary Compution, Vol. 6, No. 4, pp. 333-346.
[17] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I.Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen and R. F.Freund, (2001) “A Comparison of Eleven Static Heuristics for mappinga Class of Independent Tasks onto Heterogeneous DistributedComputing Systems”, Journal of Parallel and Distributed Computing.Vol.61, no. 6, pp.810-837.
[18] G. Ritchie and J. Levine, (2003) “A fast, effective local search for schedulingindependent jobs in heterogeneous computing environments”.Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group.UNSPECIFIED.
[19] R. Armstrong, D. Hensgen, and T. Kidd,(1998)“The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions,” in 7th IEEE Heterogeneous Computing workshop, pp. 79–87.
[20] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M.Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.Lima, F. Mirabile, L. Moore, B. Rust, and H. Siegel(1998)“Scheduling resources in multi-user, heterogeneous,computing environments with SmartNet,” in 7th IEEEHeterogeneous Computing Workshop, pp. 184–199.
[21] Li Liu, Yi Yang, Lian Li and Wanbin Shi,(2006)“Using AntOptimization for super scheduling in computational Grid",IEEE proceedings of the 2006 IEEE Asia-pasific Conferenceon Services Computing (APSCC’ 06)
[22] Bharadwaj, V., Ghose, D., Mani, V. and Robertazzi, T.G., (1996) "Scheduling Divisible Loads in Parallel andDistributed Systems", IEEE Computer Society Press, LosAlamitos CA.
[23] Dorigo M., Maniezzo, V., Colorni, A.(1996) “Ant system: Optimization by a colony of cooperating agents”. IEEE Transactions on Systems, Man, and Cybernetics. Vol 26,no. 1, pp. 29–41.