Ant Algorithm for Grid Scheduling Problem

Stefka Fidanova and Mariya Durchova

IPP – BAS, Acad. G. Bonchev, bl.25A, 1113 Sofia, Bulgaria
stefka@parallel.bas.bg, mabs@parallel.bas.bg

Abstract. Grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data storage or network resources across dynamic and geographically dispersed organizations. The goal of grid task scheduling is to achieve high system throughput and to match the application needed with the available computing resources. This is matching of resources in a non-deterministically shared heterogeneous environment. The complexity of scheduling problem increases with the size of the grid and becomes highly difficult to solve effectively. To obtain good methods to solve this problem a new area of research is implemented. This area is based on developed heuristic techniques that provide an optimal or near optimal solution for large grids. In this paper we introduce a tasks scheduling algorithm for grid computing. The algorithm is based on Ant Colony Optimization (ACO) which is a Monte Carlo method. The paper shows how to search for the best tasks scheduling for grid computing.

1 Introduction

Computational Grids are a new trend in distributed computing systems. They allow the sharing of geographically distributed resources in an efficient way, extending the boundaries of what we perceive as distributed computing. Various sciences can benefit from the use of grids to solve CPU-intensive problems, creating potential benefits to the entire society. With further development of grid technology, it is very likely that corporations, universities and public institutions will exploit grids to enhance their computing infrastructure. In recent years there has been a large increase in grid technologies research, which has produced some reference grid implementations.

Task scheduling is an integrated part of parallel and distributed computing. Intensive research has been done in this area and many results have been widely accepted. With the emergence of the computational grid, new scheduling algorithms are in demand for addressing new concerns arising in the grid environment. In this environment the scheduling problem is to schedule a stream of applications from different users to a set of computing resources to maximize system utilization. This scheduling involves matching of applications needs with resource availability.

There are three main phases of scheduling on a grid [10]. Phase one is resource discovery, which generates a list of potential resources. Phase two involves gathering information about those resources and choosing the best set to match the

I. Lirkov, S. Margenov, and J. Waśniewski (Eds.): LSSC 2005, LNCS 3743, pp. 405–412, 2006.

application requirements. In the phase three the job is executed, which includes file staging and cleanup. In the second phase the choice of the best pairs of jobs and resources is NP-complete problem [4].

A related scheduling algorithm for the traditional scheduling problem is Dynamic Level Scheduling (DLS) algorithm [11]. DLS aims at selecting the best subtask-machine pair for the next scheduling. To select the best subtask-machine pair, it provides a model to calculate the dynamic level of the task-machine pair. The overall goal is to minimize the computational time of the application. In the grid environment the scheduling algorithm no longer focuses on the subtasks of an application within a computational host or a virtual organization (clusters, network of workstations, etc.). The goal is to schedule all the incoming applications to the available computational power. In [1, 7] some simple heuristics for dynamic matching and scheduling of a class of independent tasks onto a heterogeneous computing system have been presented.

There are two different goals for task scheduling: high performance computing and high throughput computing. The former aims is minimizing the execution time of each application and later aims is scheduling a set of independent tasks to increase the processing capacity of the systems over a long period of time.

Our approach is to develop a high throughput computing scheduling algorithm based on ACO. ACO algorithm can be interpreted as parallel replicated Monte Carlo (MC) systems [12]. MC systems [9] are general stochastic simulation systems, that is, techniques performing repeated sampling experiments on the model of the system under consideration by making use of a stochastic component in the state sampling and/or transition rules. Experimental results are used to update some statistical knowledge about the problem, as well as the estimate of the variables the researcher is interested in. In turn, this knowledge can be also iteratively used to reduce the variance in the estimation of the described variables, directing the simulation process toward the most interesting state space regions. Analogously, in ACO algorithms the ants sample the problem's solution space by repeatedly applying a stochastic decision policy until a feasible solution of the considered problem is built. The sampling is realized concurrently by a collection of differently instantiated replicas of the same ant type. Each ant "experiment" allows to adaptively modify the local statistical knowledge on the problem structure. The recursive retransmission of such knowledge determines a reduction in the variance of the whole search process the so far most interesting explored transitions probabilistically bias future search, preventing ants to waste resources in not promising regions of the search.

The organization of the paper is as follows. In section 2 the ACO method is discussed. In section 3 grid scheduling algorithm is introduced. We make some experimental testing and conclude this study in sections 4 and 5.

2 Ant Colony Optimization

Real ants foraging for food lay down quantities of pheromone (chemical cues) marking the path that they follow. An isolated ant moves essentially at random but an ant encountering a previously laid pheromone will detect it and decide to

follow it with high probability and thereby reinforce it with a further quantity of pheromone. The repetition of the above mechanism represents the auto catalytic behavior of real ant colony where the more the ants follow a trail, the more attractive that trail becomes.

The ACO algorithms were inspired by the observation of real ant colonies [2, 3]. An interesting behavior is how ants can find the shortest paths between food sources and their nest. While walking from a food source to the nest and vice-versa, ants deposit on the ground a substance called pheromone. Ants can smell pheromone and then they tend to choose, in probability, paths marked by strong pheromone concentrations. The pheromone trail allows the ants to find their way back to the food source (or to the nest).

The above behavior of real ants has inspired ACO algorithm. ACO algorithm, which is a population-based approach, has been successfully applied to many NP-hard optimization problems [2, 3]. One of its main ideas is the indirect communication among the individuals of ant colony. This mechanism is based on an analogy with trails of pheromone which real ants use for communication. The pheromone trails are a kind of distributed numerical information which is modified by the ants to reflect their experience accumulated while solving a particular problem.

The ACO algorithm uses a colony of artificial ants that behave as co-operative agents in a mathematical space were they are allowed to search and reinforce pathways (solutions) in order to find the optimal ones. Solution that satisfies the constraints is feasible. After initialization of the pheromone trails, ants construct feasible solutions, starting from random nodes, then the pheromone trails are updated. At each step ants compute a set of feasible moves and select the best one (according to some probabilistic rules) to carry out the rest of the tour. The transition probability is based on the heuristic information and pheromone trail level of the move. The higher value of the pheromone and the heuristic information, the more profitable it is to select this move and resume the search. In the beginning, the initial pheromone level is set to a small positive constant value τ_0 and then ants update this value after completing the construction stage.


```
procedure ACO
begin
    Initialize the pheromone
    while stopping criterion not satisfied do
        Position each ant in a starting node
        repeat
            for each ant do
                Chose next node by applying the state transition rate
            end for
        until every ant has build a solution
        Update the pheromone
    end while
end
```

All ACO algorithms adopt specific algorithmic scheme as is shown above. After the initialization of the pheromone trails and control parameters, a main loop is repeated until the stopping criteria are met. The stopping criteria can be a certain number of iterations or a given CPU time limit or time limit without improving the result. In the main loop the ants construct feasible solutions and then the pheromone trails are updated. More precisely, partial problem solutions are seen as states: each ant starts from random state and moves from state i to another state j of the partial solution. At each step, ant k computes a set of feasible solutions to its current state and moves to one of these expansions, according to a probability distribution specified as follows. For ant k the probability p_{ij}^k to move from a state i to a state j depends on the combination of two values:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}.\eta_{ij}}{\sum_{l \in allowed_k} \tau_{il}.\eta_{il}} & if\ j \in allowed_k \\ 0 & otherwise \end{cases} \quad (1)$$

where

- η_{ij} is the attractiveness of the move as computed by some heuristic information indicating a prior desirability of that move;
- τ_{ij} is the pheromone trail level of the move, indicating how profitable it has been in the past to make that particular move (it represents therefore a posterior indication of the desirability of that move);
- $allowed_k$ is the set of remaining feasible states.

Thus, the higher the value of the pheromone and the heuristic information, the more profitable it is to include state j in the partial solution. In the beginning, the initial pheromone level is set to τ_0, which is a small positive constant. In the nature there is not any pheromone on the ground at the beginning, or the initial pheromone in the nature is $\tau_0 = 0$. If in ACO algorithm the initial pheromone is zero, than the probability to chose next state will be $p_{ij}^k = 0$ and the search process will stop from the beginning. Thus it is important the initial pheromone to be positive value.

The pheromone level of the elements of the solutions is changed by applying following updating rule:

$$\tau_{ij} \leftarrow \rho.\tau_{ij} + \Delta\tau_{ij}, \quad (2)$$

where the rule $0 < \rho < 1$ models evaporation and $\Delta\tau ij$ is an additional pheromone and it is different for different ACO algorithms. Normally the quantity of the added pheromone depends on the quality of the solution.

3 Grid Scheduling Model

Our scheduling algorithm is designed for distributed systems shared asynchronously by both remote and local users.

3.1 Grid Model

The grid considered in this study is composed of a number of hosts send, each host is composed of several computational resources, which may be homogeneous or heterogeneous. The grid scheduler does not own the local hosts, therefore does not have control over them. The grid scheduler must make best effort decisions and then submit the jobs to the hosts selected, generally as a user. Furthermore, the grid scheduler does not have control over the set of jobs submitted to the grid, or local jobs submitted to the computing hosts directly. This lack of ownership and control is the source of many of the problems yet to be solved in this area. The grid scheduling is a particular case of tasks scheduling on machines problem. In the grid scheduling every machine can execute any task, but for different time.

3.2 Grid Scheduling Algorithm

While there are scheduling request from applications, the scheduler allocates the application to the host by selecting the best match from the pool of applications and pool of the available hosts. The selecting strategy can be based on the prediction of the computing power of the host [6]. We will review some terms and definitions [7, 8].

The expected execution time ET_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given that m_j has no load when t_i is assigned. The expected completion time CT_{ij} of the task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let M be the total number of the machines. Let S be the set containing the tasks. Let the beginning time of t_i be b_i. From the above definitions, $CT_{ij} = b_i + ET_{ij}$. The makespan for the complete schedule is then defined as $\max_{t_i \in S}(CT_{ij})$. Makespan is a measure of the throughput of the heterogeneous computing system. The objective of the grid scheduling algorithm is to minimize the makespan. It is well known that the problem of deciding on an optimal assignment of jobs to resources is NP-complete. We develop heuristic algorithm based on ACO to solve this problem.

Existing mapping heuristics can be divided into two categories: on-line mode and batch mode. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, tasks are not mapped onto the machines as they arrive, instead they are collected in a set that is examined for mapping at pre-scheduled times called mapping events. This independent set of tasks that is considered for mapping at mapping events is called meta-task. In the on-line mode, each task is considered only once for matching and scheduling. The minimum completion time heuristic assigns each task to the machine so that the task will have the earliest computation time [5]. The minimum execution time heuristic assigns each task to the machine that performs that tasks' computation in the least amount of execution time. In batch mode, the scheduler consider a meta-task for matching and scheduling at each mapping event. This enable the mapping heuristics to possibly make better decision, because the heuristics have the resource requirement information for the meta-task and known the actual

execution time of a larger number of tasks. Our heuristic algorithm is for batch mode.

Let the number of the tasks in the set of tasks is greater than the number of machines in the grid. The result will be triples $(task, machine, startingtime)$. The function $free(j)$ - shows when the machine m_j will be free. If the task t_i is executed on the machine m_j then the starting time of t_i becomes $b_i = free(j)+1$ and the new value of the function $free(j)$ becomes $free(j) = b_i + ET_{ij} = CT_{ij}$.

An important part of implementation of ACO algorithm is the graph of the problem. We need to decide which elements of the problem to correspond to the nodes and which ones to the arcs. Let $M = \{m_1, m_2, \ldots, m_m\}$ is the set of the machines and $t = \{t_1, t_2, \ldots, t_s\}$ is the set of the tasks and $s > m$. Let $\{T_{ij}\}_{s \times m}$ is the set of the nodes of the graph and to machine $m_j \in M$ corresponds a set of nodes $\{T_{kj}\}_{k=1}^{s}$. The graph is fully connected. The problem is to choose s nodes of the graph thus to minimize the function $F = max(free(j))$, where $[b_i, CT_{ij}] \cap [b_k, CT_{kj}] = \oslash$ for all $i,\ j,\ k$. We will use several ants and every ant starts from random node to create their solution. There is a tabu list corresponding to every ant. When a node T_{ij} is chosen by the ant, the nodes $\{T_{ik}\}_{k=1}^{m}$ is included in tabu list. Thus we prevent the possibility the task t_i to be executed more than ones. An ant add new nodes in the solution till all nodes are in the tabu list. Like heuristic information we use:

$$\eta_{ij} = \frac{1}{free(j)}.$$

Thus if a machine is free earlier, the corresponding node will be more desirable. At the end of every iteration we calculate the objective function $F_k = max(free(j))$ over the solution constructed by ant k and the added pheromone by the ant k is:

$$\Delta\tau_{ij} = \frac{(1-\rho)}{F_k}.$$

Hence in the next iterations the elements of the solution with less value of the objective function will be more desirable. Our ACO implementation is different from ACO implementation on traditional tasks machines scheduling problem. The new of our implementation is using of multiple node corresponding to one machine. It is possible because in grid scheduling problem every machine can execute any task.

Two kind of sets of tasks are needed: set of scheduled tasks and set of arrived and unscheduled tasks. When the set of scheduled tasks becomes empty the scheduled algorithm is started over the tasks from the set of unscheduled tasks. Thus is guaranteed that the machines will be fully loaded.

4 Experimental Testing

We have developed 3 simulated grid examples to evaluate the newly proposed ACO algorithm for grid scheduling. In our experimental testing we use 5 heterogeneous machines and 20 tasks. The initial parameters are set as follows:

$\tau_0 = 0.01$ and $\rho = 0.5$ and we use 1 ant. We compare achieved by ACO algorithm result with often used online-mode.

The results are in minutes. We observe the outperform of ACO algorithm and the improvement of the result with. In online-mode the arriving order is very important. In ACO algorithm the most important is the execution time of the separate task.

Table 1. Makespan for the execution on first free machine and ACO algorithm

online-mode	ACO	improvement
80	67	16%
174	128	26.4%
95	80	15.8%

5 Conclusion

To confront new challenges in tasks scheduling in a grid environment, we present in this study heuristic scheduling algorithm. The proposed scheduling algorithm is designed to achieve high throughput computing in a grid environment. This is a NP-problem and to be solved needs an exponential time. Therefore the heuristic algorithm which finds a good solution in a reasonable time is developed. In this paper heuristic algorithm based on ACO method is discussed and it basic strategies for a grid scheduling are formulated. This algorithm guarantee good load balancing of the machines. In ACO technique it is very important how the graph of the problem is created. Another research direction is to create different heuristic based algorithms for problems arising in grid computing.

Acknowledgments

Stefka Fidanova was supported by the European Community program "Structuring the European Research Area" contract No MERG-CT-2004-510714. Mariya Durchova was supported by the Bulgarian IST Center of Competence in 21^{st} century — BIS-21++ funded by European Commission in FP6 INCO via grant 016639/2005.

References

1. Braun T. D., Siegel H. J., Beck N., Bolony L., Maheswaram M., Reuther A. I., Robertson J.P., Theys M. D., Jao B.: *A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems*, IEEE Workshop on Advances in Parallel and Distributed Systems, (1998) 330–335.
2. Dorigo, M., Di Caro, G.: The Ant Colony Optimization metaheuristic. In: *New Idea in Optimization*, Corne, D., Dorigo, M., Glover, F. eds., McGrow-Hill (1999) 11–32.

3. Dorigo, M., Gambardella, L.M.: Ant Colony System: A cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transaction on Evolutionary Computation*, **1** (1999) 53–66.
4. Fernandez-Baca D.: Allocating Modules to Processors in a Distributed System, *IEEE Transactions on Software Engineering*, **15** 11 (1989) 1427–1436.
5. Freund R. F., Gherrity M., Ambrosius S., Camp-Bell M., Halderman M., Hensgen D., Keith E., Kidd T., Kussow M., Lima J.D., Mirabile F., Moore L., Rust B., Siegel H.J.: Scheduling Resources in Multi-User Heterogeneous Computing Environments with SmartNet, IEEE Heterogeneous Computing Workshop, (1998) 184–199.
6. Gong L., Sun X.H., Waston E.: Performance Modeling and Prediction of Non-Dedicated Network Computing, *IEEE Transaction on Computer*, **51** 9 (2002) 1041–1055.
7. Maheswaran M., Ali S., Siegel H.J., Hensgen D., Freund R.: *Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems*, 8th IEEE Heterogeneous Computing Workshop (HCW'99), San Juan, Puerto Rico, (1999) 30–44.
8. Pinedo M.:*Scheduling: Theory, Algorithms and Systems*, Prentice Hall, Englewood Clifts, NJ, (1995).
9. Rubinstein, R. Y.:*Simulation and the Monte Carlo Method*. John Wiley &Sons, (1981).
10. Schopf J.M: *A General Architecture for Scheduling on the Grid*, special issue of JPDC on Grid Computing, (2002).
11. Sih G.C., Lee E.A.: A Compile-Time Scheduling Heuristic for Inter Connection-Constrained Heterogeneous Processor Architectures, *IEEE Transactions Parallel and Distributed Systems*, **4** (1993) 175–187.
12. Strelsov, S., Vakili, P.: Variance Reduction Algorithms for Parallel Replicated Simulation of Uniformized Markov Chains. *J. of Discrete Event Dynamic Systems: Theory and Applications*, **6** (1996) 159–180.