A Dynamic Task Scheduling Algorithm for Grid Computing System

Yuanyuan Zhang[1], Yasushi Inoguchi[2], and Hong Shen[1]

[1] Graduate School of Information Science,
[2] Center for Information Science,
Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292, Japan
{yuanyuan, inoguchi, shen}@jaist.ac.jp

Abstract. In this paper, we propose a dynamic task scheduling algorithm which assigns tasks with precedence constraints to processors in a Grid computing system. The proposed scheduling algorithm bases on a modified static scheduling algorithm and takes into account the heterogeneous and dynamic natures of resources in Grid.

1 Introduction

Grid computing[1], the internet-based infrastructure that aggregates geographically distributed and heterogeneous resources to solve large-scale problems, is becoming increasingly popular. Heterogeneity, dynamicity, scalability and autonomy are four key characteristics of Grid.

A critical issue for the performance of Grid is that of task scheduling. Task scheduling problem is, in general, the problem of scheduling tasks to processors so that all the tasks can finish their execution in the minimal time. Since this problem is NP-complete in general, it is necessary to employ heuristics to arrive at a near-optimal solution.

Many task scheduling heuristics have been proposed for heterogeneous system [2, 3, 4], however, most of these algorithms can't work for Grid directly because resources in Grid are typically heterogeneous and the performance of the resources dynamically fluctuates over time as the resources are not dedicated to Grid.

In this paper, we propose a new scheduling algorithm which assigns precedence-constrained tasks to the processors in a Grid computing system. Our algorithm takes the dynamicity and heterogeneity of Grid into consideration so that it can better deal with the dynamics of dynamically varying resource state in Grid.

The rest of this paper is organized as follows: We formalize the problem and describe the proposed algorithm in Section 2 and Section 3 concludes the paper.

J. Cao et al. (Eds.): ISPA 2004, LNCS 3358, pp. 578–583, 2004.

2 The Proposed Algorithm

2.1 Problem Statement

An application to be executed on the Grid system consists of many precedence-constrained tasks. The application is represented by a Directed Acyclic Graph in which every node represents a task of the application, and each directed edge represents a communication(comm. in short) link between two tasks. Edge $e_{i,j}$ represents the dependence relationship between task T_i and task T_j that T_i must finish before T_j can start.

For edge $e_{i,j}$, we call T_j a *successor* of T_i, and T_i a *predecessor* of T_j. The size of data communicated from T_i to T_j is $data_{i,j}$. A task is called an entry task if it has no predecessor, while a task is called an exit task if it has no successor. We assume that there is only one entry task and one exit task in the application.

The communication cost between task T_i and T_j, denoted by $C_{i,j}$, is the expected transfer time of the data for the communication given current network conditions. For this purpose, Network Weather Service(NWS) can be used to obtain an estimate of the current network latency and bandwidth. It is assumed that the communication cost on a processor is neglectable.

Let τ_i denote the workload of Grid task T_i, and υ_m be the speed of processor P_m, then τ_i/υ_m is the time to implement T_i on P_m when P_m is dedicated to execute T_i. However, the actual speed of P_m delivered to Grid is less than υ_m and varies over time since the resource owner also uses it and the local jobs have higher priority over the Grid tasks. This may dramatically impact the performance of Grid resources, and makes the problem more difficult.

The objective function is to schedule the tasks of the application to the processors in Grid so that to minimize the total execution time.

2.2 The Proposed Task Scheduling Algorithm

Most static scheduling algorithms assume that task execution times and data transfer times can be estimated accurately and are commonly used with dedicated resources. However, the fact in Grid is that it is often difficult to obtain such accurate estimation before execution. Therefore, dynamic scheduling algorithms may have better performance on Grid because they are able to adapt to environmental dynamicity. In this paper we propose a new dynamic algorithm to deal with the dynamic and heterogeneous features of Grid.

First we must solve the nondedicated feature of Grid: during the implementation of a Grid task on a processor, the local jobs on the processor will arrive and interrupt the implementation of the Grid task. We consider the execution of the local jobs as non-preemptive, i.e., a local job must run until completion once it starts. The execution of the local jobs follows the rule of *first-come first-serve*. From the viewpoint of the Grid task, the state of the processor alternates between available and unavailable: when the processor is executing its own jobs, it's unavailable for the Grid task, otherwise it's available for the Grid task.

If we assume the arrival of the local jobs in processor P_m follows a Poisson distribution with arrival rate λ_m, and their execution process follows an expo-

nential distribution with service rate μ_m, then the local job process in P_m is an M/M/1 queuing system.

The expected execution time $\omega^e_{i,m}$ of task T_i on P_m can be expressed as:

$$\omega^e_{i,m} = X^m_1 + Y^m_1 + X^m_2 + Y^m_2 ... + X^m_{N_m} + Y^m_{N_m}, \tag{1}$$

where N_m is the number of local jobs which arrive during the execution of T_i, and X^m_j, Y^m_j(j = 1, ..., N_m), are respectively the computing time of a section of the Grid task and a local job. Y^m_j(j = 1, 2, ..., N_m) are independent identical distribution(i.i.d.) random variables. We have:

$$X^m_1 + X^m_2 + ... + X^m_{N_m} = \tau_i/\upsilon_m. \tag{2}$$

From the knowledge of queuing theory, we have:

$$E(N_m) = \lambda_m \tau_i/\upsilon_m, \qquad E(Y^m_j) = 1/(\mu_m - \lambda_m). \tag{3}$$

Since N_m and Y_j are independent(j = 1, ..., N_m), we can derive:

$$\mathrm{E}(\omega^e_{i,m}) = E(E(\omega^e_{i,m}|N_m)) = \tfrac{\tau_i}{\upsilon_m(1-\rho_m)}. \tag{4}$$

where $\rho_m = \lambda_m/\mu_m$, is the utilization rate of P_m.

For a processor P_m with utilization rate ρ_m, we can use $\frac{\tau_i}{\upsilon_m(1-\rho_m)}$ as the expected execution time of T_i on P_m. However, ρ_m is a value that reflects the dynamicity of the Grid during a long time. It can not reflect the dynamicity during the execution of the application, therefore we introduce the concept of *processor credibility* which reflects the history of prediction accuracy for a processor during the execution of a Grid application. We denote the credibility of P_m as δ_m, which has the original value of 1 when we schedule the application. After a task T_i finishes execution on P_m, we can obtain its actual execution time $\omega^a_{i,m}$ and so that δ_m is modified as:

$$\delta_m = (1-\alpha)\delta_m + \alpha * \omega^a_{i,m}/\omega^e_{i,m}, \tag{5}$$

where α is a value between 0 and 1 and can be modified.

Therefore the expected execution time $\omega^e_{i,m}$ of T_i on P_m is modified as:

$$\omega^e_{i,m} = \frac{\tau_i \delta_m}{\upsilon_m(1-\rho_m)}. \tag{6}$$

The *average execution time* of T_i, which is denoted by ω_i, is defined as:

$$\omega_i = \sum_{m=1}^{q} \omega^e_{i,m}/q = \sum_{m=1}^{q} \frac{\tau_i \delta_m}{\upsilon_m q(1-\rho_m)}, \tag{7}$$

where q is the number of processors in Grid.

The dynamic algorithm we propose bases on a static algorithm which is a modified Heterogeneous Earliest Finish Time(HEFT) algorithm[3]. HEFT is a

traditional task scheduling algorithm for precedence-constrained tasks in heterogeneous systems. The scheduling process of this algorithm includes two phases: task selection and processor selection. In task selection phase, the tasks are queued by non-increasing ranks. For task T_i, its rank $rank_u(T_i)$, is computed as:

$$rank_u(T_i) = \omega_i + \max_{T_j \in succ(T_i)} (C_{i,j} + rank_u(T_j)). \tag{8}$$

$$\text{Here} \qquad C_{i,j} = L + \frac{data_{i,j}}{B}. \tag{9}$$

In the above equation $succ(T_i)$ is the set of the successors of T_i. L and B are respectively the *average communication startup time* and *average bandwidth* among all the processors. $rank_u$ of each task is computed by traversing all the tasks upward, starting from the exit task whose $rank_u$ is defined as:

$$rank_u(T_{exit}) = \omega_{exit}. \tag{10}$$

In processor selection phase, the first task in the list is selected and allocated to the processor which gives it the minimal earliest finish time(EFT).

For task T_i, let EST(T_i,P_m) and EFT(T_i,P_m) denote its *earliest start time* and *earliest finish time* on processor P_m. AFT(T_i) is its *actual finish time.* avail[P_m] denotes the time when P_m is ready for executing new tasks. We also denote the *comm. finish time* between T_i and its successor T_j as CFT(T_i,T_j).

In the HEFT algorithm, when computing EFT(T_i,P_m) of T_i on P_m, the algorithm computes CFT(T_j,T_i) between T_i and its predecessor T_j which has not been scheduled to P_m, and determines EST(T_i,P_m) as the maximum value between CFT(T_j,T_i) and avail[P_m]. If we have decided to schedule T_i on processor P_m, and if avail[P_m] is less than CFT(T_j,T_i), then P_m will remain idle when it is waiting for the finish of communication between T_j and T_i. However, if we adopt the idea of task duplication scheme in this algorithm, that is, when the communication time is long compared with computation time, we duplicate T_j which has been scheduled to other processors on P_m, thus the communication time will be removed, so that the total execution time of the tasks will be probably shortened. We name this modified HEFT algorithm as Duplication-based HEFT(DHEFT) algorithm.

The performance of DHEFT depends heavily on the granularity of communication between the tasks. For communication-intensive applications, it can achieve much better performance than HEFT algorithm, otherwise its advantage will be trivial. So we use DHEFT algorithm only for communication-intensive Grid application, otherwise we use the original HEFT algorithm. In our future work the threshold of comm.-comp. ratio for judging if an application should be scheduled using DHEFT or not will be determined.

In our dynamic scheduling algorithm, at first we relate every task with a level so that tasks with same level are independent. The level of a task is defined as the length of the longest path from the entry task to the present task. By the length of a path we mean the number of tasks on the path. The level of

T_{entry} is 1. After we schedule T_{entry} to a processor, we begin its execution immediately. Then when tasks with level i-1 start execution(i>1) we dynamically schedule tasks with level i using DHEFT algorithm for communication-intensive tasks or HEFT algorithm. When i>2, since tasks with level less than i-1 have finished execution when we schedule tasks with level i , we use the estimation error of the finished tasks to modify the credibility of every processor using equation (6), so that to regulate the estimated execution time of the tasks in level i .

The following pseudocode shows the implementation process of DHEFT algorithm and the dynamic scheduling algorithm:

Duplication-based HEFT Algorithm (DHEFT)	**Dynamic Task Scheduling Algorithm**
Sort the tasks in list Q_1 by decreasing $rank_u$	Compute the level of every task
While there are tasks remained in Q_1 {	EST(T_{entry}) = 0
Select the first task T_i from Q_1	For every processor P_m
For processor P_k { t = avail[P_k]	avail[P_m] = 0
If all the predecessors of T_i are on P_k or	$Q_1 \leftarrow$ set of tasks whose level is 1
T_i is the entry task	Compute $rank_u$ of every task in Q_1
EFT(T_i, P_k) = t + $\omega^e_{i,k}$	Schedule the tasks in Q_1
Else	Start the execution of tasks in Q_1
EST(T_i, P_k) = t	$Q_1 \leftarrow$ set of tasks with level 2
Sort predecessors of T_i which are not	Compute $rank_u$ of tasks in Q_1
on P_k in Q_2 by decreasing $rank_u$	Schedule the tasks in Q_1
Select the first task T_j from Q_2	For i = 2; i < level(T_{exit}); i++;
If EFT(T_j) + $C_{j,i} \leq$ t + $\omega^e_{j,k}$	Wait until any task in level i begins
EST(T_i, P_k) = MAX(t, EFT(T_j) + $C_{j,i}$)	execution
Else	Refresh the credibility of every
t = t + $\omega^e_{j,k}$	processor using actual execution
EST(T_i, P_k) =t	times of tasks in level i-1
EFT(T_i, P_k) = EST(T_i, P_k) + $\omega^e_{i,k}$}	$Q_1 \leftarrow$ set of tasks with level i+1
Allocate T_i to P_m which gives it the minimal	Compute $rank_u$ of tasks in Q_1
EFT(T_i, P_m)	Schedule the tasks in Q_1
avail[P_m] = EFT(T_i,P_m) }	

3 Conclusion

In this paper, we propose a task scheduling algorithm for Grid computing system. The proposed scheduling algorithm considers the dynamicity and heterogeneity of Grid. It combines the list scheduling and task duplication scheme. It also uses the execution results of the finished tasks to regulate the expected execution times of the tasks to be scheduled, so that to obtain more accurate estimation and arrive at better scheduling result.

References

1. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Fransisco, CA, 1999.
2. C. Banino, O. Beaumont, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor grids," Applied Parallel Computing: Advanced Scientific Computing: 6th Int'l Conf., pp. 423-432, Jun. 2002.
3. H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, Mar. 2002.
4. J.-C. Liou and M.A. Palis, "An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors," Workshop on Resource Management, Symposium on Parallel and Distributed Processing. 1996.