社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004−ARC−157 (20)
2004−HPC− 97 (20)
2004／3／2

Implementation and Evaluation of an Adaptive Scheduling Method for a Grid Computing Environment

Georgi GEORGIEV*, Kalin KOZHUHAROV†, Mitsuyoshi NAGAO‡, Manabu OMIYA‡

* Faculty of Engineering, Hokkaido University
† Graduate School of Engineering, Hokkaido University
‡ Information Initiative Center, Hokkaido University

Abstract Grid Computing is a next generation method of information processing. It provides an effective method for information processing by connecting geographically distributed computer resources using a high-speed network and providing standard protocol for communication. A scheduling method is required in order to implement an effective method for computation in the grid computing environment, because each resource has different characteristics, e.g., calculation speed. In this paper, we propose an adaptive scheduling method for effective information processing in a grid computing environment that consists of diverse computer resources. Moreover, we confirm its effectiveness through computational experiments in practical grid computing environments.

グリッド環境のための適応的スケジューリング法の実現と評価

ゲオルギエヴ ゲオルギ* コジュハロフ カリン† 長尾 光悦‡ 大宮 学‡

* 北海道大学工学部
† 北海道大学大学院工学研究科
‡ 北海道大学情報基盤センター

あらまし グリッドコンピューティングは，次世代の新たな情報処理手法の一つである．これは，物理的に分散した計算機リソース群を結合することによって効果的な情報処理を可能とする手法である．しかしながら，グリッドコンピューティング環境における計算機リソース群は演算速度等の性能が異なる。したがって、これらリソース群に基づき有効な情報処理を実現するためには，効果的なスケジューリング方法が必要とされる．本稿では，多種多様な計算機リソースが結合されたグリッドコンピューティング環境において効果的な情報処理を実現するための適応的スケジューリング法を提案する．さらに，計算機実験を通じて提案手法の有効性を検証する．

1. Introduction

Recently, grid computing has been applied to various research fields. It is an information processing method for large-scale problem solving. Grid computing is expected to become an effective technology for solving a wide range of problems—from earthquake simulations to teleimmersion.

In grid computing, physically distributed computer resources are connected by a high speed network. Conventional research has focused on grid computing environments that are constructed by connecting mainly super computers or clusters [1] [2]. Some scheduling methods that are suitable for such grid computing environments have been proposed [3] [4].

On the other hand, recently the performance of personal computers (PC) is being improved. It is likely that a more effective grid computing environment can be implemented by introducing PCs. However, if a grid computing environment consists of computer resources that differ extremely in performance, there is the possibility that effective performance cannot be achieved by using conventional scheduling methods.

In this paper, we propose an adaptive scheduling method in order to implement effective information processing in a grid computing environment consisting of diverse computer resources. In this research, we focus on a scheduling method for parametric applications. The proposed scheduling method aims at finding the optimal allocation of parametric jobs over the avail-

able computer resources. The allocation is determined by predicting the running time of every parametric job on every resource. This prediction is performed by executing the application inside the grid computing environment that is to be used for the calculation before practical use.

In addition, the grid computing environment is used by many users and the condition of resources, i.e., CPU load, in the environment varies frequently. There are cases where our scheduling method does not work effectively when such a change of condition occurs, because there is a difference in the load between the condition when the predictions are made and when the application is executed. To solve this problem, we improve our method by introducing a function which can adapt to the changes in the grid computing environment. We confirm the effectiveness of the proposed scheduling method through some practical computational experiments at various conditions in a grid computing environment.

2. Grid Computing

2.1 Overview

Grid computing is a technology that uses distributed computer resources connected by a high-speed network for information processing. The spreading of high-speed network infrastructures made the grid a frequently researched field [5] [6].

The NASA Information Power Grid (IPG) is a typical example of a grid computing implementation. The IPG consists of high-performance computers and clusters in three different research centers in the U.S. The purpose of this project is to promote various research by providing a powerful grid computing environment.

Currently, grid computing is still an incipient research topic. The middleware and high-level tools for grid computing are under development.

2.2 Globus Toolkit

The most popular middleware for constructing a grid computing environment is the globus toolkit [7]. It is a collection of primitive tools necessary for this task. In addition, it is open source and runs on various platforms such as Linux, Unix, IRIX and so on. The current version of the globus toolkit is 3.0. Globus toolkit version 3.0 employs SOAP technology in order to solve the firewall problems that version 2.4 has. The globus toolkit has become the de-facto standard for grid computing.

The basic components of the Globus toolkit are the Grid Resource Allocation Manager (GRAM), GridFTP and the Monitoring and Discovery Service (MDS). GRAM takes care of resource reservation and submits jobs to the job-manager. It is also responsible for allowing control and monitoring over the execution of the job. GridFTP is a service that provides secure, safe and high-performance file transfer. MDS is the directory service of the globus toolkit. It allows for the discovery of resources, retrieving information about their characteristics and monitoring their state. A grid computing environment can be implemented by combining these components.

2.3 Parametric Application

Parallel applications can benefit a lot from grid computing, because the grid is a distributed computing system. Applications designed to run on massively parallel computers can easily be run in a grid computing environment that is based on the globus toolkit, because the appropriate libraries such as MPICH-G2 are provided [8].

Another type of applications suitable for grid computing is the parametric applications. A parametric application is made from a single program and a set of different parameters that the program is to be run with. The complete computational result for a parametric application is obtained by merging each result acquired from every run of the application with a different parameter. A parametric application can be executed in a grid computing environment without the need for specific libraries. The most important characteristic of the parametric application is that the different jobs, i.e., the runs of the application with different parameters, are independent of one another and can be freely scheduled, without paying attention to the interprocess communication factor. In this research, we focus on the parametric applications.

2.4 Scheduling

In the grid computing environment, a scheduler is required for effective information processing. A scheduler is needed to assign appropriate jobs to appropriate resources, so that the application is executed the way the user demands. So far, some schedulers have been developed.

Nimrod-G is a scheduler, that supports its own scripting language for describing the experiment and the individual jobs [3]. It has support for cost and speed optimizations and submits jobs to the available resources that are fit for the purpose. It also provides a graphical user interface for monitoring and administering the scheduling process.

AppLeS is mentioned as another project [4]. It is based on the idea that a scheduler has to be written specifically for the application or type of application being run, i.e., there are no schedulers that are fit for every kind of application. Each specific task, such as selecting resources, developing scheduling plans or evaluating their performance, is performed in a different module.

The Network Weather System (NWS) has been proposed as a supporting system for the schedulers [10]. It provides short-term prediction about the future state of a computer resource in a grid computing environment. It can provide information about the available CPU, CPU load, free memory and so on. The scheduler can allocate appropriate jobs to each computing resource by using this information. The NWS is used by many conventional schedulers including the schedulers described above. However, the information provided by the NWS is not always accurate, because the NWS cannot consider what type of job is executed on the computing resource. Therefore, there are cases where effective scheduling cannot be performed if it is based on the NWS.

3. Adaptive Scheduling Method for a Grid Computing Environment

We aim to provide an effective scheduler for parametric applications for grid computing. Our grid environment and the details of our proposed method are described below.

3.1 Grid Environment

Our grid environment is illustrated on Figure 1. As shown in this figure, it includes computer resources with various specifications. Two middle-class personal computers and a parallel computer SGI Onyx 300 are included as computing resources for our grid computing environment. In addition, another PC acts as the task broker, i.e. the scheduler. The PCs that act as computing resources have Pentium III/850MHz and Pentium III/1GHz CPUs. The SGI Onyx has 32 nodes running at 600MHz each and the broker has an Athlon/1GHz CPU. The SGI Onyx is running the IRIX6.5 operating system, and the other resources are running a Linux 2.4 based system. All PCs have version 2.4 of the globus toolkit installed, and the SGI Onyx has version 2.2. The PCs are connected to a 1Gbps hub, but only have 100Mbps network interface cards. The connection between the hub and the SGI Onyx is 1Gbps.

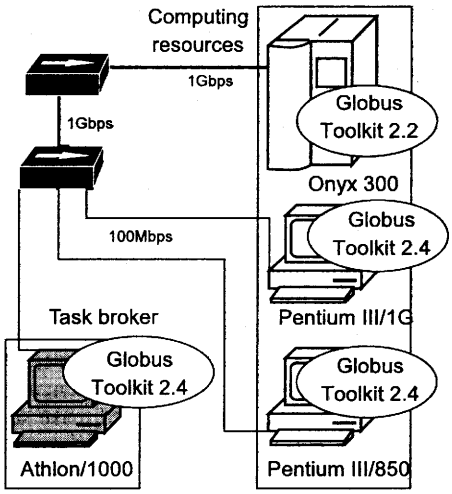

Figure 1 Grid environment

3.2 Adaptive Scheduling Method

The basic concept of the proposed method is that the parametric jobs of the application are wrapped in meta-jobs equal exactly to the number of available computer resources. In other words, each computing resource is only used once, i.e., only one job is submitted to a computing resource. In this method, it is important that all computing resources finish calculating at the same time.

A lot of procedures such as authentication, authorization and negotiation with the job manager are required when a job is submitted to a computing resource, which creates a large overhead. In conventional schedulers, the user decides the size of each job, and then all these jobs are submitted to computing resources sequentially. When there are many jobs, the overhead becomes large and the execution becomes time consuming. Our proposed scheduling method decreases the overhead by minimizing the number of jobs.

In the case where the application is divided into many small jobs, a new job is submitted to the computing resource that finishes execution of its previous one. Thus, the calculation time does not depend greatly on the performance of the computing resources. However, if the application is simply split in a number of jobs equal to the total number of computing resources, without predicting how much computation each of the jobs would require, a fast computing resource may receive a job that will be executed quickly, while the more intensive jobs go to slower resources. Even

if the jobs are equally computation-intensive, the total execution time would depend on the speed of the slowest computing resource. To avoid these problems, in the proposed method, the contents of the meta-jobs are selected after predicting the performance of each job on each resource. The goal is to create such meta-jobs that they all finish at the same time. The details of the performance prediction are as follows.

(1) A time function f that represents a relation between the parameters and the running time has to be found. However, because the parameter is not necessarily a number, a function g that gives the representation of the parameter s as a number has to be found. The function should make sure that if $g(s_1) = g(s_2)$, then when s_1 or s_2 is passed as a parameter the running time of the application is the same. Therefore, to determine a proper function $g(s)$, the source code of the application may have to be examined.

(2) The general form of the time function f can be obtained by investigating the running time. This general form is the same regardless of the computer resource that the application is executed on. However, the complete function varies between computer resources. The exact parameters of the time function for each computer resource are determined by executing a sufficient number of parametric jobs on each computer resource in the grid computing environment.

For example, if f is a function of a second degree, i.e.,

$$f(x) = ax^2 + bx + c \qquad (1)$$

jobs respective to only three points have to be executed on each computer resource. The values for a,b and c can then be determined from their running time.

(3) After the function f for each resource is acquired, a distribution of jobs is determined. This is done according to the following equation

$$\sum_{i}^{\sigma_1} f_1(g(i)) = \sum_{i}^{\sigma_2} f_2(g(i)) = \cdots \sum_{i}^{\sigma_n} f_n(g(i)) \qquad (2)$$

where σ_i is the set of parameters assigned to the i^{th} resource. The application is executed on the basis of these parameters.

The proposed scheduling requires the execution of a job on the computing resources in order to perform accurate performance prediction of the resource. Therefore, the proposed method is not suitable for the case when the application is not utilized repeatedly, because the calculation time for the prediction is added to total calculation cost.

This scheduling method works effectively in a grid environment that consists of parallel computers or super computers only or when the computer resources have no extra load. Parallel computers have multiple calculation nodes. The job is basically assigned to an unused calculation node when a new job is submitted to a parallel computer. Thus, a job can monopolize the utilization of the CPU. However, this is not possible if there are other processes on a single processor system. Processes equally share the CPU.

In the proposed method, the performance of the computing resources is measured, the predictions are made, and then the real application is executed. Therefore, the performance of the proposed method could drastically decrease if the conditions on a computing resource when performing the prediction differ from the conditions when executing the application. To avoid this problem, we introduce adaptability to the proposed scheduling method. This way, the method can be applied in more practical grid computing environments.

The number of active processes is checked when measuring the running time of the application. The time that it would take to complete on a single processor system is directly proportional to the total number of competing processes; i.e. if the time when the CPU is dedicated to the application is t_0, then the time when there are n competing processes is $t_n = (n+1)t_0$. Therefore, if the number of running processes when performing the prediction measurements is p_p and the measured time is t, $\frac{t}{(p_p+1)}$, i.e., the time that would have been measured if there were no other processes, is used when determining the parameters in Equation (1).

When determining the distribution of jobs, $(p_e + 1)f(g(s))$ is used instead of $f(g(s))$ in Equation (2), where p_e is the number of active processes at the time of execution of the real job. This change of the equation creates a different distribution of jobs, respecting the current load.

As mentioned above, the proposed scheduling method can perform effective execution of a parametric application by minimizing the overhead of job submission, choosing a job size based on performance prediction of the computing resources

and adjusting the job size on the basis of the current load.

4. Computational Experiments

4.1 Experimental Method

We performed computational experiments in order to confirm the effectiveness of the proposed scheduling method. The experimental methods are described below.

The application used in these experiments was a program for calculating arbitrary hexadecimal digits of π [9]. Its implementation is written in C. This application accepts as parameters the starting digit and the number of digits for calculation. In this program, later digits require more calculation. In the experiment, the first one hundred thousand hexadecimal digits of π were calculated. We also compared our proposed scheduling method with the conventional scheduling methods described below.

(1) **Scheduling Method 1 (M1):** The application is split in jobs of equal size (equal number of digits) and the number of jobs is equal to the number of computing resources.

(2) **Scheduling Method 2:** The application is divided in a number of fixed-size jobs, but the number of jobs is much greater than the number of computing resources. We employed two different job sizes, because job size also has impact on performance.

- **1000 digits ($M2_a$):** A thousand digits of π are calculated by each job, i.e., the application is split in one hundred jobs.
- **2500 digits ($M2_b$):** The application is divided into forty jobs that calculate 2500 digits each.

The above experiments were performed in ideal environment, i.e., the computer resources had no load. Moreover, we performed the following types of experiments in order to confirm the effectiveness of the adjustment function based on the load condition in our proposed scheduling method:

- **Experiments with no extra load:** In the experiments there was no extra load on the PCs both during prediction and real execution.
- **Experiments with varying load:** In the experiment, there was some external load on the PC resource and it was different between the prediction and the real execution of the application.

The load used in the experiments is controlled. A fixed load is applied when the load generator program is executed on a computing resource.

In these experiments, we investigated the elapsed time of execution.

4.2 Experimental Results

The experimental results are shown in Tables 1 and 2. Table 1 represents the experimental results in the case where there was no load, and Table 2 represents the results using varied load. The data is the average of ten trials.

In Table 1, M1 is when the application is simply divided into equally sized pieces. In other words, this is our method without the prediction. From the results, it can be concluded that this is the most unpredictable method. This is due to the fact that the jobs in this method have different computational requirements, but they are not distributed to faster/slower computer resources according to these requirements. However, at given runs even this method did complete the job faster than any of the M2 methods. Therefore, even though this method is not predictable, when the right selection of resources is made, the method is faster than M2.

By comparing the two M2 methods, the impact of the job-submission overhead can be seen. $M2_a$ that had 100 jobs was slower than $M2_b$ that had 40. Furthermore, there is no easy way of deciding in advance what the job size should be. We decided to work with job sizes of 1000 and 2500 after performing a few test runs and seeing that these sizes give the best results. With a job-size of 5000 the results were becoming unstable, because at a number of runs the last job had to be waited for.

Our proposed method showed both stable and fast results. In other words, there were no unexpectedly slow executions and it was always the faster method of the three. The results are stable, unlike M1. Therefore, it can be concluded that the prediction is effective. It performed better than the M2 methods, which means that keeping the job in large pieces is effective. However, with the proposed method, the prediction runs and determining the time function for the application take time. This extra time is not required for M1 or M2. On the other hand, if the application is used frequently, this extra time becomes negligible when compared to the gain in speed that it provides. It is our future work to decrease this extra time.

Table 1 Accomplished times with no extra load

Resources			Time (sec.)			
			M1	$M2_a$	$M2_b$	Prop.
O*	$P_1^\dagger$	$P_2^\ddagger$	1363~3187	1477	1596	1246
O	P_1		1828~3741	1822	2135	1597
O		P_2	1820~4261	1966	2661	1680
	P_1	P_2	3612~4269	2851	3037	2561

*Onyx 300, †P/III 1GHz, ‡P/III 850MHz

Table 2 Accomplished times with varying load

Load at:	execution			
measure	0	1	2	3
0	1598	1912	2052	2140
1	1615	1910	2044	2150
2	1636	1924	2050	2120
3	1605	1952	2062	2127

Table 2 represents the results when the load on the computer resource during prediction differs from the load when the application is executed. From this result, it was found that the remarkable decrease of performance in the proposed method did not occur even if the load condition had changed between prediction and real execution. If there were no adjustments and the load on one computer resource had changed, the calculation time in Table 2 would have been as follows. If the load at execution was l_e, i.e., there were l_e processes running on the computer resource already, and the load when predicting was l_p, then if $l_e < l_p$ the running time t_{l_e} would have been the same as the time when the load is l_p, i.e., t_{l_p}. However, if $l_e > l_p$, then the time would have been $t_{l_e} = t_{l_p} \frac{l_e+1}{l_p+1}$. It can therefore be concluded that the adjustments were effective.

However, the load for the experiments was artificially generated by a program that provided constant load. However, in a more practical environment, there are different types of load. We have to confirm the performance of the proposed method when loads differ. This is our future work.

5. Conclusion

In this paper, we proposed an adaptive scheduling method for the grid computing environment. We also confirmed its effectiveness through some computational experiments. From the experimental results, it was found that the proposed scheduling method can effectively manage the execution of parametric applications by predicting the performance on each computing resource and by adjusting the job size on the basis of load conditions.

The proposed method requires extra time for predicting the performance on each computing resource. Therefore, the conventional scheduling methods can be more effective when a completely unknown parametric application is executed for the first time. However, it is likely that the proposed method will become superior if the time required for making the predictions can be decreased.

In addition, it was confirmed that the proposed scheduling method also worked effectively even if the load on a computer resource has changed. However, in the experiment, we simulated the change of the condition on a computer resource by applying a constant load generated by a program. Therefore, we also have to investigate the performance of the proposed method through experiments with various load types. This is our future work.

Bibliography

[1] http://www.nas.nasa.gov/About/IPG/ipg.html

[2] R.Rheinheimer, S.L.Humphries, H.P.Bivens, J.I.Beiriger, The ASCI computational Grid: Initial Deployment

[3] D.Abramson, J.Giddy, L.Kotler, High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, IPDPS'2000, Mexico, IEEE CS Press, USA, 2000

[4] F.Berman, R.Wolski. The AppLeS Project: A Status Report. 8th NEC Research Symposium, Berlin, Germany, May 1997. 16

[5] The Anatomy of the Grid: Enabling Scalable Virtual Organizations. I.Foster, C.Kesselman, S.Tuecke. International J. Supercomputer Applications, 15(3), 2001.

[6] Computational Grids., I. Foster, C. Kesselman. Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 1999.

[7] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing, SIAM, Lyon, France, August 1996

[8] MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. N. Karonis, B. Toonen, and I. Foster. Journal of Parallel and Distributed Computing, 2003.

[9] D.H.Bailey, D.Broadhurst,Y.Hida and Xiaoye S. Li, "High Performance Computing Meets Experimental Mathematics", 2002

[10] The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, R. Wolski, N. Spring, J. Hayes, Journal of Future Generation Computing Systems, Volume 15, Numbers 5-6, pp. 757-768, October, 1999.