Performance Comparison of Process Allocation Schemes Depending upon Resource Availability on Grid Computing Environment

Hiroshi Yamamoto[1], Kenji Kawahara[1], Tetsuya Takine[2], and Yuji Oie[1]

[1] Dept. of Computer Science and Electronics, Kyusyu Institute of Technology, Kawazu 680-4, Iizuka, 820-8502 Japan
yamamoto@infonet.cse.kyutech.ac.jp
{kawahara, oie}@cse.kyutech.ac.jp
[2] Dept. of Applied Mathematics and Physics Graduate School of Informatics, Kyoto University, Yoshidahonmachi, Sakyou–ku, Kyoto, 606–8501 Japan
takine@amp.i.kyoto-u.ac.jp

Abstract. Improvements in the performance of end-computers and networks have recently made the construction of a grid system over the Internet feasible. A grid environment comprises many computers, each having a set of components and distinct performance, that are shared among many users and managed in a distributed manner. Thus, it is important to focus on a situation in which the computers are used unevenly due to decentralized management by different process schedulers. In the present study, as a preliminary theoretical investigation of the effect of such features on the performance of process schedulers, the average execution time of a long-lived process in the process allocation scheme employed in the decentralized environment is analytically derived using M/G/1–PS queues. The impact of the distribution of CPU utilization on the performance of these schemes is also investigated assuming CPU utilization for each computers in a grid environment follows a probability distribution function.

1 Introduction

Recent improvements in the performance of end-computers and networks have made it feasible to construct a grid system over the Internet. A grid system [1] is constructed by connecting geographically distributed computers over the Internet to secure greater computing power. Several fundamental services are indispensable for sharing computational resources (e.g., CPU time, memory, storage, etc.) in a grid environment, including security management, resource management, and process execution on remote hosts, which are run as a low-level middleware [2]. The Globus Toolkit [3] produced by the Globus project [4] is well-known middleware providing these important services. Using these services, a computer (User) can securely submit a set of process executions to the

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2658, pp. 247–256, 2003.

grid environment. Users require a scheduling system linked to the middleware to appropriately select computers to which the processes are to be allocated.

The grid environment consists of many computers, each with distinct processing performance, and the resources of each computer are shared among many users who are geographically distributed in a decentralized manner. The resource status of each computer is managed in a distributed manner rather than a centralized manner, most likely by many different organizations [5]. Although the number of processes executed on each computer and its utilization cannot be controlled strictly, the resources can be used unevenly and dynamically. To achieve efficient grid computing, it is necessary to develop a process allocation scheme that considers a distributed resource management architecture. Process allocation schemes based upon the resource status of available computers have already been proposed [6][7]. However, such schemes do not cater for decentralized resource management, instead assuming that users are able to perceive all processes, like in cluster computing.

In the present research, we focus particular attention on a situation in which computers are very likely to be used unevenly and utilization will change dynamically because of decentralized management by different process schedulers. Specifically, in this paper, the effect of such features on the performance of process schedulers is preliminarily investigated based on theoretical considerations. The process allocation schemes treated here are implemented in a decentralized manner in which users initially investigate resource status such as CPU performance and utilization, and then divide local long-lived process into a number of *sub-processes* to be allocated to computers with good resource availability.

The computers are modeled as M/G/1–PS (processor sharing) queues, and the average execution time of long-lived process in each scheme are derived analytically. Adopting M/G/1–PS queues allows the average sub-process sojourn time as a function of resource status alone. As the grid computing environment is constantly evolving, it is very difficult to precisely describe the CPU utilization. Hence, a probability distribution function associated with CPU utilization is introduced to describe the uneven and dynamic utilization of computers, allowing the impact of these features on the performance of proposed process allocation schemes to be evaluated.

The rest of this paper is organized as follows: In Section 2, a resource management model of grid computing is described, and the process allocation schemes are proposed. Section 3 outlines the analytical evaluation of the average process execution time, which is used in section 4 to evaluate the characteristics of the proposed process allocation schemes. Finally, this paper is concluded in section 5.

2 Grid Computing Architecture

2.1 Resource Management Architecture

The resource management architecture employed here is shown in Fig. 1, consisting of computers and directories.

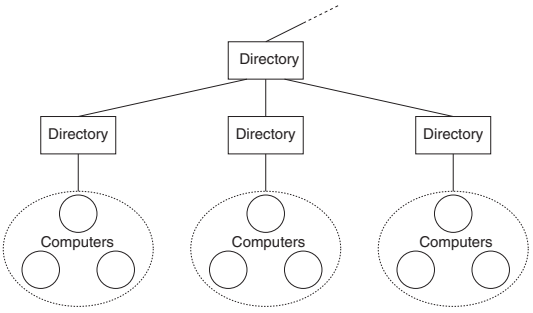

Fig. 1. Resource Management Architecture.

Computers Every computer is assumed to manage its own performance and capability using a database. This information is measured periodically using a resource management toolkit such as the Network Weather Service [8]. In this context, computers execute processes allocated by other computers involved in the grid, while simultaneously executing necessary local processes.

Here, CPU performance is denoted C_i, and CPU utilization ρ_i $(0 \leq \rho_i < 1)$ is defined as the load imposed on the computer R_i. The computer can execute processes allocated to it at the rate of C_i per unit time, and ρ_i of the total capability of the CPU is consumed when the user allocates new sub-process.

Directory The directory manages the static computer information as a database, including the operating system, processing capability, memory capacity, and IP addresses. The database is updated periodically when computers notify the directory. According to requests from users, the directory asks computers under its control to send back information related to the resource status, which is cached in the directory database, and then replies to users with the obtained information. Each entry registered with a database has a lifetime [9]: if the entry has not been updated during the lifetime, the directory determines that the relevant computer is no longer available.

2.2 Grid Computing

The grid computing process is outlined in Fig. 2. After a process arrives, the user divides long-lived processes into n independent small-scale sub-processes. The long-lived process has a computational complexity of X. Here, as overhead in each sub-process due to segmentation is neglected for simplicity, the computational complexity of each sub-process x can be defined as X/n. Next, the user requests the directory to investigate the resource status of computers, and to reply with the relevant information. The user selects n computers to which sub-processes are allocated based on the information from the directory. The computer resource information is presented in detail in section 3. In this study,

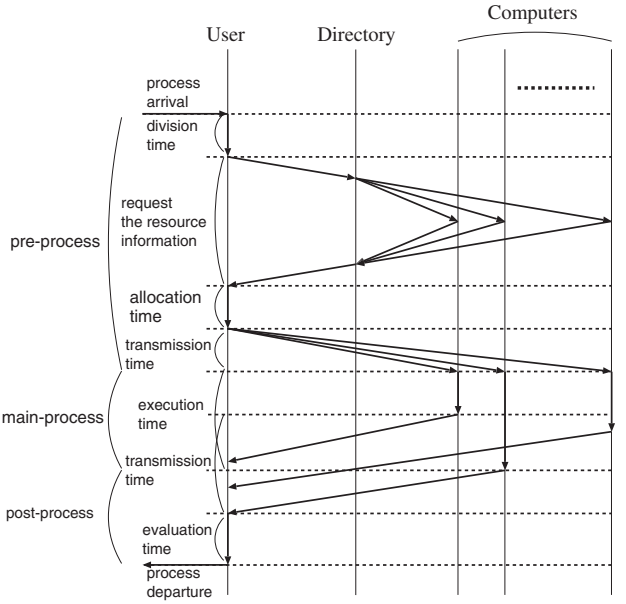

Fig. 2. Grid Computing.

it is reasonably assumed that all processes are independent, have the same priority, and are fairly scheduled in a round-robin manner on all computers. The overall process execution time is determined by the time when the result of the latest sub-process is returned.

As shown in Fig. 2, the interval from the allocation of sub-processes to the completion of all sub-processes is defined as the *main-process*. Processes to be executed before the main-process begins are defined as the *pre-process*, and those executed after the main-process are defined as the *post-process*. In this paper, the performance of each sub-process allocation scheme is evaluated based on the process execution time for the main-process.

2.3 Sub-process Allocation Scheme

Some adaptive sub-process allocation schemes are described below. Here, the limited case of at most one sub-process allocated to each computer is considered.

Scheme 1 (Random Selection). The user randomly selects n (number of sub-processes) computers from all the available computers, then allocates all n sub-processes as selected. In this scheme, the user does not have to consider the computer resource information, thereby minimizing the pre-process time.

Scheme 2 (Resource Availability First Selection). The user selects n least-loaded computers and allocates sub-processes to these computers. In this scheme,

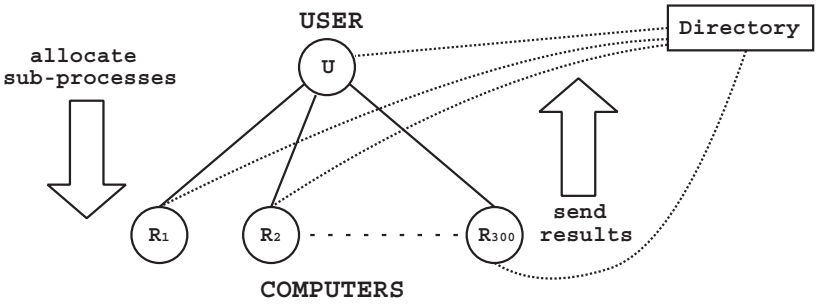

Fig. 3. Analytical Model.

the user needs to receive the current resource information for all computers and compare the available processing capability. This results in a lengthy pre-process time.

Scheme 3 (Random Selection with Resource Availability Threshold). The user requests the directory to ask all computers to send back a reply if the computer satisfies the requirement that the current utilization is lower than some threshold T. The user then randomly selects n computers among those responding to the inquiry, and allocates sub-processes accordingly. When T is set relatively low, fewer than n computers may satisfy the requirement, in which case the user increases T by some amount and re-investigates the resource status. This process is repeated until n or more computers satisfy the requirement.

The pre-process time of Scheme 3 is expected to be shorter than that of Scheme 2 because the user does not need to receive resource information from all available computers, and also does not need to compare the information.

3 Analytical Model

As illustrated in Fig. 3, the analytical model is defined such that the user is directly connected to 300 computers over the Internet. Transmission delay between users is neglected. The performance of each of the schemes described in section 2.3 is evaluated using this model. Computers are evenly divided into three groups in terms of CPU performance; class A, B and C, corresponding to CPU performance C_A, C_B and C_C of 50, 100 and 150, respectively. A CPU performance of 1.0 indicates that the computer takes a unit time of 1.0 to calculate a process of unit workload. Thus a CPU with a performance of 50 can calculate a process with a workload of 100 in 2 unit periods. As only the main-process is considered here, the average process execution time is adopted as a performance measure, and is obtained analytically as follows.

Each computer is modeled using M/G/1–PS queues, where all processes in the system are fairly served with a round-robin scheduler. Thus, this model allows for the coexistence of other processes, which may be allocated from other

users and/or generated by the local computer. If a sub-process with computational complexity x is allocated on computer R_i with CPU performance C_i and utilization ρ_i, the sub-process mean sojourn time is given by

$$E[W_i] = \frac{x/C_i}{1-\rho_i}. \tag{1}$$

Considering that the average process execution time is defined as the interval between the allocation and receipt of all sub-process, if one process is divided by the user into n sub-processes, which are allocated to computers $R_1, \cdots, R_n$ the average process execution time is defined as the maximum sub-process mean sojourn time as follows.

$$E[W] = \max\{E[W_1], \cdots, E[W_n]\}. \tag{2}$$

The current resource capability I_i of computer R_i, which indicates the average processing time for a process with unit workload, is defined as

$$I_i = \frac{1/C_i}{1-\rho_i}. \tag{3}$$

Even if the CPU performance of the three groups differ, the user can select computers in order of the value I_i to allocate sub-processes in Scheme 2. In Scheme 3, the user randomly selects computers with current resource capability I_i higher than the predetermined threshold T. Thus, each scheme mentioned in section 2.3 is valid, and is evaluated as follows:

1. Computer resource information is investigated using Eq. (3).
2. Sub-processes are allocated to n computers according to the current resource capability.
3. The average process execution time is evaluated using Eq. (2).

However in general, the CPU utilization changes dynamically. Therefore, 10,000 experiments were conducted under the condition that the average CPU utilization of available computers $\bar{\rho}$ $(0 \leq \bar{\rho} < 1)$ is fixed while the CPU utilization of each computer is changed. The mean of the average process execution time for all experiments is then employed as the performance measure.

The grid computing environment is constantly evolving, which makes it very difficult to precisely describe the CPU utilization of computers involved. Thus, the CPU utilizations is assumed to follow a power distribution in which the CPU utilization ρ_i of computer R_i is given as follows.

$$F(\rho_i) = \rho_i^c. \qquad \left(i = 1, \cdots, 300, \quad c = \frac{\bar{\rho}}{1-\bar{\rho}}\right) \tag{4}$$

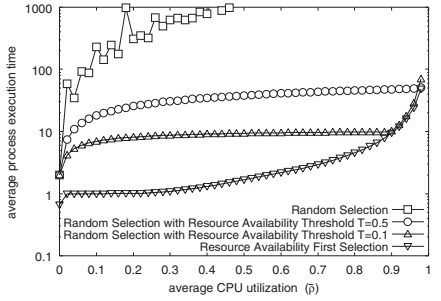

Fig. 4. Average Process Execution Time vs. Average CPU Utilization.

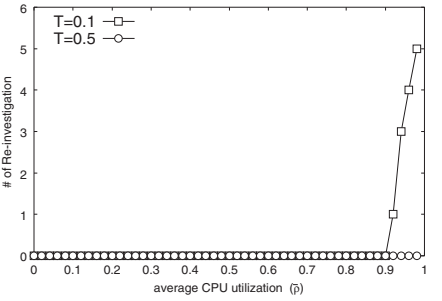

Fig. 5. Number of Re-investigations vs. Average CPU Utilization.

4 Results and Discussion

4.1 Average Process Execution Time vs. Average CPU Utilization

The workload of each process X (i.e., computational complexity) was set to 10,000, and the number of sub-processes n was set to 100. Therefore, the computational complexity x of each sub-process is 100 (=10000/100).

Figure 4 shows the average process execution time as a function of average CPU utilization. In random selection (Scheme 1), 100 computers are randomly chosen independent of CPU utilization or class. Therefore, some of the sub-processes will be assigned to computers of class A, resulting in an execution time of as long as 2 (=100/50) even when the average CPU utilization is 0. With the power distribution, allocation under scheme 1 may result in assignment to a highly loaded computer. As a result, the average process execution time increases markedly with average CPU utilization.

In Scheme 2, the user selects the n least-loaded computers and allocates sub-processes accordingly. Consequently, the average process execution time of Scheme 2 outperforms the other schemes treated here. The performance of Scheme 3 falls between schemes 1 and 2 in that n computers are selected randomly, but from computers that are not highly loaded. From Fig. 4, as the threshold T is reduced, the average process execution time becomes smaller over a wide range of $\bar{\rho}$ and approaches that of Scheme 2. However, reducing T may also result in re-investigation if too few computers are available. Although this is a concern, Fig. 5 shows that no re-investigation occurs under the present model until $\bar{\rho}$ becomes more than 0.9, even in the case of $T = 0.1$.

4.2 Ratio of Sub-processes Allocated to Computers in Each Class

Given the present method of updating computer information, it may occur that many users will select similar sets of computers based on information that may not be the latest, which could lead to critical performance degradation.

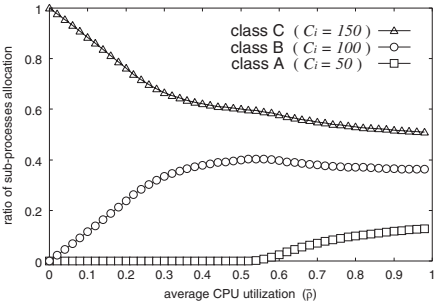

Fig. 6. Ratio of Sub-Processes Allocated to Each Class (Resource Availability First Selection).

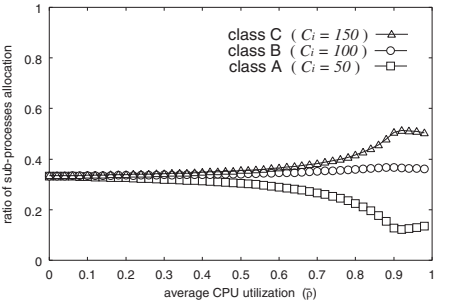

Fig. 7. Ratio of Sub-Processes Allocated to Each Class (Random Selection with Resource Availability Threshold: $T = 0.1$).

To examine this in more detail, the ratio of sub-processes allocated to computers belonging to each class is investigated. As shown in Fig. 6, Scheme 2 allocates many sub-processes to computers with higher CPU performance (i.e., class C). As the average CPU utilization increases, computers with relatively low CPU performance are also selected. However, even if the average CPU utilization is 0.5, 60% of all sub-processes are assigned to computers in class C. Therefore, it is very likely under Scheme 2 that many users in the grid environment will select a few computers with high performance or that were lightly loaded at the same time, thereby leading to critical performance degradation if processes are allocated independently by several schedulers.

Figure 7 shows that Scheme 3 selects computers evenly from all classes. For example, in the case that the average CPU utilization is smaller than 0.5, the ratio of sub-processes allocated to each class is around 0.33. Therefore, Scheme 3 will use all types of computers fairly, while providing moderate performance as shown in Fig. 4.

4.3 Effect of the Number of Sub-processes on the Average Process Execution Time

The effect of the number of sub-processes is illustrated in Fig. 8 in terms of the average process execution time for average CPU utilization $\bar{\rho}$ of 0.6. Normally, as the number of sub-processes increases, the complexity of each sub-process x decreases, resulting in a comparable decrease in average process execution time for Scheme 2 down to some minimum value. As shown in Fig. 8, as the number of sub-processes is increased past 130, the average process execution time begins to increase again because the user must allocate sub-processes to highly loaded computers with higher probability. The average process execution time for Scheme 3 improves with the number of sub-processes as long as all the selected computers have I_i smaller than T. From Fig. 8, the average process

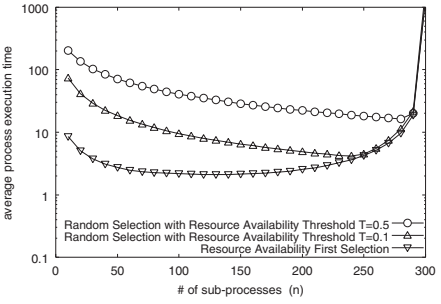

Fig. 8. Average Process Execution Time vs. the Number of Sub-processes.

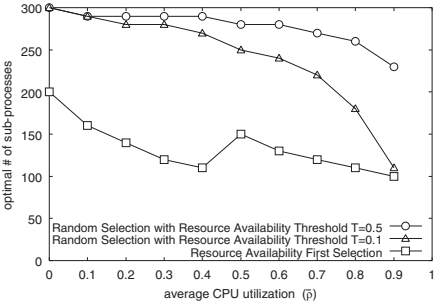

Fig. 9. Optimal Number of Sub-Processes.

execution time decreases with increasing n for $T = 0.1$ as long as n is smaller than 240. In this case, no re-investigation occurs.

As mentioned above, there exists an optimal number of sub-processes that minimizes the average execution time for some value of $\bar{\rho}$. Figure 9 illustrates the optimal number of sub-processes for each $\bar{\rho}$. The optimal number of sub-processes is dependent on $\bar{\rho}$, demonstrating that a larger number of sub-processes does not always lead to an improvement in the process execution time. Therefore, the number of sub-processes should be determined carefully in consideration of $\bar{\rho}$, that is, the average load imposed on the grid computing system. This figure also shows that the optimal number of sub-processes is larger for higher threshold T.

5 Conclusion

The performance of sub-process allocation schemes was analytically evaluated using a model grid computing environment in which computers were modeled as M/G/1–PS queues. Three process allocation schemes were examined; random selection of computers (Scheme 1), selection of n least-loaded computers based on the current resource availability (Scheme 2), and random selection of n computers from among those with processing capability greater than a predefined threshold (Scheme 3). The latter two schemes employ current resource availability information for computers in the grid, requiring a model of CPU utilization. To achieve this in a grid computing environment, a power distribution was used to describe an environment in which computers are very likely to be used unevenly and to have dynamically changing utilization.

The performance of each scheme was investigated preliminarily in terms of the average process execution time expressed as a function of average CPU utilization. It was also shown that there exists an optimal number of sub-processes at which grid computing is most effective. Scheme 1 is very simple to implement, but the user may unknowingly choose computers that are highly loaded. Scheme 1 is therefore considered to be inadequate even if the average CPU utilization is

relatively low. Scheme 2 outperforms both of the other schemes treated here, but may concentrate sub-process allocation to a few computers, possibly resulting in sever performance degradation if processes are independently allocated by several process schedulers at a similar time. Scheme 3 achieves the most moderate performance over a wide range of average CPU utilization without concentrating process allocation. Therefore, Scheme 3 appears to be greatest practical interest in terms of ease of implementation and moderate performance provided. The optimal number of sub-processes was shown to vary dependent on average CPU utilization, indicating that a larger number of sub-processes does not always lead to an improvement in the process execution time.

Acknowledgments This work was supported in part by a Grant-in Aid for Scientific Research on Priority Areas (2) (14019074) of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. I. Foster and C. Kesselman, *The GRID Blueprint for a New Computing Infrastructure,* Morgan Kaufmann Publishers, 1998.
2. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid", *International Journal of Supercomputer Applications*, Vol. 15, Num. 3, 2001.
3. I. Foster and C. Kesselman, "The Globus Project: A Status Report", *Proc. of IPPS/SPDP '98 Heterogeneous Computing Workshop*, pp. 4–18, 1998.
4. The Globus Project, *http://www.globus.org/.*
5. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", *Proc. of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC–10)*, IEEE Press, August 2001.
6. E. Heymann, M. A. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for Master-Worker Applications on the Computational Grid", *Proc. of the First International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, December 2000.
7. G. Shao, F. Berman, and R. Wolski, "Master/Slave Computing on the Grid", *Proc. of 9th Heterogeneous Computing Workshop*, Cancun, Mexico, pp. 3–16, May 2000.
8. R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Journal of Future Generation Computing Systems*, Vol. 15, Num. 5–6, pp. 757–768, October 1999.
9. S. Gullapalli, K. Czajkowski, C. Kesselman, and S. Fitzgerald, "Grid Notification Framework (VB0.1)", *Grid Forum Working Draft GWD-GIS-019, www.gridforum.org*, June 2001.