An introduction to load balancing for parallel raytracing on HDC systems

Kalim Qureshi* and Masahiko Hatanaka

Department of Computer Science and Systems Engineering., Muroran Institute of Technology, Mizumoto cho 27-1, Muroran, 050-8585 Hokkaido, Japan

Heterogeneous distributed computing (HDC) systems, which exploit the aggregate power of a network of workstations and personal computers are an inexpensive alternative to the dedicated parallel supercomputing systems. As these systems are widely available in academic and industrial environments, it is becoming popular to use these computing resources to solve time-consuming applications.

The focus of this paper is to empirically identify the load balancing problems in parallel raytracing on HDC system and present a short survey of current load balancing schemes for HDC systems.

1. Introduction

RAYTRACING is a method of generating realistic computer images. For many applications it requires a large amount of computation due to calculations of colour and intensity for each pixel by tracing light rays[1]. Moreover, often the computation requirements and application data size cross the limits of a single workstation (WS)/personal computer (PC).

The advent of commodity-based distributed computing environments has made it possible to process such kind of images in reasonable time on modest budgets. The main problem with such type of cluster computing environments is the continuous change in the performance of individual (node) WS/PC, which requires effective task partitioning, scheduling, and load balancing to get better performance.

Many researchers suggest enhancements in adaptive task scheduling strategies for homogeneous distributed system[2–4]. However, these strategies do not work well for heterogeneous distributed computing (HDC) systems without further modifications. The crucial point is that they are based on fixed parameters that are tuned for the specific hardware. In HDC systems, this tuning is often not possible because both the computational power and the network bandwidth are not known in advance – they may change unpredictably during run-time.

Strategies based on task distribution and task migration from heavily loaded to lightly-loaded nodes are discussed in Sarje and Sagar[5]. The task migration has two serious drawbacks[6].

i) All nodes should continuously monitor the status of other nodes.
ii) During the computations, a node has to identify its load and float the information on the network, hence, this produces a large amount of communication overhead.

This paper is organized as follows. In next section, we describe the fundamental classification of load balancing schemes for HDC systems and description of investigated strategies. Strategy performance evaluation metrics and the experimental set-up are explained in section 3. Section 4 presents the obtained results and discussions. Finally, the conclusion of this paper and future research directions are given in section 5.

2. Load balancing schemes for HDC

One of the biggest issues in HDC systems is how to distribute and schedule tasks among computing-resources/nodes to have some performance goal, such as minimizing execution time, minimizing communication delays, and/or maximizing resources utilization[7].

Scheduling techniques can be classified based on the availability of program task information as *deterministic* and *non-deterministic*. In deterministic scheduling, the information about tasks to be scheduled and their relations to one another is entirely known prior to execution time. In non-deterministic scheduling, some information may not be known before the execution of the program. Both non-deterministic and deterministic scheduling can be implemented using static, dynamic, or hybrid methods.

In static scheduling, the assignment of the tasks to the nodes is done before the execution of the program. Information regarding task execution time and processing resources is assumed to be known at compile time. A task is always executed on the node to which it is assigned.

Dynamic scheduling is based on the re-distribution of processes among the processors during execution time. This redistribution is performed by transferring tasks from

*For correspondence. (e-mail: qur@wave.csse.muroran-it.ac.jp)

heavily-loaded processors to lightly-loaded processors with an aim to minimize the processing time of the application. The load balancing operations may be centralized in a single processor or distributed among all the processing elements that participate in the load balancing process. Many combined policies may also exit. For example, the information policy may be centralized but the task transfer and placement policies may be distributed. Interested readers may refer to refs 7–9 for additional details.

The advantage of dynamic load balancing over static scheduling is that the system need not be aware of run-time behaviour of the application before execution. The flexibility inherent in dynamic load balancing allows for adaptation to unforeseen application requirements at run-time. The major disadvantage of dynamic load balancing schemes is the run-time overhead due to:

- The load information transfer among processors,
- The decision-making process for the selection of processes and processor for job transfers, and
- The communication delay due to task relocation itself.

2.1 *Resources estimation policies*

A resources estimation policy merely provides an infrastructure for exchange of nodes' state information. Classification of resources management policies is described below:

Centralized: A central node collects state information and constructs an estimate of the system state. The central node may be a globally shared file that is accessed and updated by all nodes. This organization has an advantage that it incurs low overhead during estimation[4]. The disadvantages are poor responsiveness of a central resource in a large-scale system resulting in poor scalability and the failure-prone nature of a central resource.

Decentralized: In a decentralized organization each node of a distributed system is responsible for collecting state information and obtaining an estimate of the system state. This type of organization has higher availability in the presence of failures, but it can potentially incur large overhead to maintain accurate state information and therefore is not easily scalable to a large-scale distributed computing system[10].

Hybrid: A hybrid organization combines both centralized and decentralized organizations, inherits their properties, and attempts to extract advantages of both organizations. A hybrid organization may be implemented in two ways. In the first case, nodes are divided into clusters and state information is exchanged within and between clusters. Membership within clusters may be decided by various factors such as network proximity, type of service performance by nodes, etc. A cluster-based hybrid scheme has shown potential for providing the desired performance in large-scale distributed computing system[11].

2.2 *Run-time task scheduling strategy*

Since WSs/PCs have performance variation characteristics[12], static task distribution is not effective for HDC systems[13]. In this investigation our aim is to quantify the load-balancing problem for raytracing application on HDC systems, using run-time task scheduling (RTS) strategy with fixed sub-task size.

In the RTS strategy, a unit of sub-task is distributed at run-time. As the node completes the previous assigned sub-task, a new task is assigned for processing. The unit of task in the RTS strategy is fixed at one horizontal scan-line of the image. If the unit of task becomes shorter, for example one pixel, it increases the inter-process communication overhead[14].

3. Performance evaluation

The HDC system used in our investigation is composed of seven Sun WSs loaded with SunOS/Solaris, and thirteen PCs (Intel-based machines) loaded with Linux/FreeBSD operating system; all of these machines are connected via Ethernet. The network communication is handled by *Open Consortium Remote Procedure Call* (ONC-RPC) *library and XDR filters*. UNIX heavy weight process technique is used to control many nodes at the manager's end. We used four distributed raytracing images that are called as scenes A, B, C and D, and each image is composed of 840×640 pixels. The five HDC configurations are set, i.e. the HDC system is composed of number of nodes (NN) = 20, 16, 12, 8 and 4. The performance of the RTS strategy is mainly evaluated in terms of speedup and nodes' idle time cost. These terms are defined briefly.

3.1 *Speedup*

The fastest machine's processing time taken to process the application individually divided by processing time taken while processing in the PDP system environment.

3.2 *Nodes' idle time cost*

(Node's new task starting time) subtracted by (Node's previous task completion time).

$$\text{Nodes' idle time cost} = \sum_{i=1}^{NT} o_i,$$

Table 1. Image scene processing times for the single fastest machine in the network

Image scene A	Image scene B	Image scene A	Image scene D
63 sec	122 sec	152 sec	244 sec

Table 2. Measured average speedup, number of requests made by the client machine to distribute task, and average nodes' idle time cost for five HDC configurations in RTS strategy

HDC system (no. of nodes)	Average speedup	Number of requests made by the client machine to distribute task	Average nodes' idle time cost (sec)
4	2.5	640	32
8	3.4	640	42
12	4.0	640	47
16	4.4	640	53
20	4.9	640	59

where NT is the total number of nodes in the current HDC configuration.

4. Results and discussions

The time taken to process the image scenes A, B, C and D by the single fastest machine in the network is shown in Table 1. All measurements are carried out when no other user has logged in to the network.

We evaluated the performance of the RTS strategy using the unit of task (one horizontal scan-line of the image). Due to the small task size, a large number of master requests occurred (see Table 2) and high overhead of nodes' idle time cost is generated. The average measured nodes' idle time cost increases as the number of nodes increases in HDC system configurations, which is due to the following reasons:

i) As the number of nodes increases, the waiting child processes increase at the master, which increases the auxiliary work load at the master machine, therefore, it may decrease its task paralellization capability and effectively increase the node's waiting time.
ii) As the number of nodes increases in the master and workers/nodes HDC system model, the low bandwidth network usage increases, because each node has the responsibility to report the results to the master. Therefore, this creates extra load on the network, which may be the cause of long nodes' idle time cost.

In RTS strategy, since the tasks are assigned to the node at run-time, the number of tasks processed by the node is proportional to the node's performance and it has a potential to absorb the machine's performance variation characteristics and non-homogeneous nature of the application. However, RTS strategy performance depends upon the size of the sub-task. If the sub-task size is too small then it generates a serious inter-process communication overhead. If the sub-task size is too large then it may create a load imbalance due to the inappropriate sub-task size of nodes, particularly for slow performance node[15].

5. Conclusions

In this paper we studied the performance of the RTS strategy for raytracing application on HDC systems. The performance can be improved by reducing the number of requests and replies made by the client machine to assign task and collect data from the node.

The following points are suggested to improve the performance of the RTS strategy:

i) Task assignment should be adaptive because the HDC system consists of unequal-performance machines. The adaptive sub-task sizes reduce the number of requests made by the client to distribute the whole task and effectively reduce the nodes' idle time cost.
ii) The effective load balancing can be achieved by task migration from slow performance nodes to higher performance nodes and by ensuring that no single node is kept idle until the full application is processed.

1. Alan Heirich and James Arvo, *J. Supercomput.*, 1998, **12**, 57–68.
2. Lee, C. and Hamid, M., *Parallel Comput.*, 1995, **21**, 137–160.
3. Reinhard, E. and Jansen, F., *Parallel Comput.*, 1997, **23**, 873–885.
4. Zhou, S., *IEEE Trans. Software Eng.*, 1988, **14**, 1327–1341.
5. Sarje, A. and Sagar, G., *IEE Proceedings-E*, 1991, **5**, 313–318.
6. Dandamudi, S., *IEEE Concurrency*, July–September 1998, pp. 63–72.
7. Casavant, T. L. and Kuhl, J. G., *IEEE Trans. Software Eng.*, 1988, **14**, 141–154.
8. Rotothor, H. G., *IEE Proc. Comput. Digit. Technol.*, 1994, **141**, 1–11.
9. Yung Terng Wang and Robert, J. T. Morris, *IEEE Trans. Comput.*, 1985, **34**, 205–217.
10. Zhou, S. and Ferrari, D., Technical Report UCB/CSD 87/336, University of California, Berkeley, January 1987.
11. Zhou, S., Zheng, X., Wang, J. and Delisle, P., Technical Report CSRI-257, University of Toronto, Toronto, Canada, April 1992.
12. Hamid, M. and Lee, C., *Parallel Comput.*, 1997, **22**, 1477–1492.
13. Zhang, Y., Kameda, H. and Hung, S. L., *IEE Proc. Comput. Digit. Technol.*, 1997, **144**, 100–107.
14. Freisleben, B., Hartmann, D. and Kielmann, T., Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences, 1997, pp. 596–605.
15. Kalim Qureshi and Masahiko Hatanaka, *Trans. IEE Jpn*, 2000, **120**, 151–157.