A Model for Dynamic Adaptive Coscheduling

LU Sanglu (~[~t~]~), ZHOU Xiaobo (~[~]~) and XIE Li (~ _~_)
National Laboratory of Novel Computer Software Technology
Department of Computer Science, Nanjing University, Nanjing 210093, P.R. China
Received August 28, 1997; revised January 4, 1998.

Abstract This paper proposes a dynamic adaptive coscheduling model DASIC to take advantage of excess available resources in a network of workstations (NOW). Besides coscheduling related subtasks dynamically, DASIC can scale up or down the process space depending upon the number of available processors on an NOW. Based on the dynamic idle processor group (IPG), DASIC employs three modules: the coscheduling module, the scalable scheduling module and the load balancing module, and uses six algorithms to achieve scalability. A simplified DASIC was also implemented, and experimental results are presented in this paper, which show that it can maximize system utilization, and achieve task parallelism as much as possible.

Keywords coscheduling, load balancing, migration, scalable design

1 Introduction

Coscheduling (or gang scheduling) has been getting popular, since Ousterhout proposed that related subtasks should be scheduled to execute simultaneously in 1982[1]. Many studies demonstrate that locally-scheduling, by which each workstation schedules the parallel applications independently, leads to unacceptable execution time for frequently-communicating processes while coscheduling may provide significant performance improvement[2].

In [3], Dussean et al. also proposed a distributed scheduling policy for parallel workloads, which focused on how to time-share between competing jobs on the same processor. Based on barrier, this algorithm can achieve dynamic implicit coordination by spin-waiting and priority-raising. Through simulation of a small subset of synthetic parallel applications (bulk-synchronous programming model), they found that the performance of implicit scheduling is near that of coscheduling (+/-35%). However, it introduces complexity and additional overhead for local scheduler, especially while making choice of appropriate spin-time for better coordination performance competing with coscheduling.

Zhang et al. of HPCS at W&M have done some researches on NOWs. Related to parallel scheduling, a self-coordinated local scheduling for parallel tasks was proposed by Du[4,5], which schedules its parallel task and local user jobs independently based on a static power preservation in that workstation so that parallel tasks on different workstations are executed at the same pace to achieve a global coordination. The method can guarantee performance for both local task and parallel task, depending on specifying a ratio at which the power is used for local and parallel workloads by local user. That cannot be well adjusted to adapt to dynamic system, e.g. it would be inefficient while the specified static power allocation ratio cannot well satisfy the power requirement desired for parallel and local workloads in practice.

As for coscheduling, all processes of one parallel application are scheduled simultaneously, with coordinated time-slicing between different applications. Coscheduling allows the communication mechanism to proceed at full speed by ensuring that processes are available for interactions when needed. It also reduces the number of context swaps arising from blocked processes and thereby reduces operating system overhead[1].

This work was supported by the Climbing Program of China and the Telecommunication Advantage Org. of Japan (TAO).
Ousterhout discussed three algorithms for coscheduling: Matrix, Continuous, and Undivided algorithms in [1]. They can all guarantee that related subtasks are scheduled to execute simultaneously. However, the number of processors in the system is unchangeable. And the number of processes of an application is also limited by assuming that no application contains processes more than the number of processors in the system. And also it does not allow dynamic process migration. Furthermore, the process slots will be in bad order with applications finishing and entering after a certain time. Among them, the Matrix is the simplest and straightforward method with fast allocation and scheduling, and the overall performance is not much worse than the other two algorithms and implementation is the simplest.

Recently coscheduling has also been proposed as a possible way to take advantage of excess idle processors on an NOW. Atallah et al. presented the Static algorithm for coscheduling computing-intensive tasks on an NOW, which tries to use the optimal subset of workstations which may involve rejecting a bad workstation even if it is available while waiting for a good workstation to become free[6]. In [7], Efe developed several coscheduling algorithms and tested their performance by simulation, including the Max, the Grab, the Static and the Freelist. The Max algorithm attempts to use as many workstations as possible at the cost of waiting possibly a long time while some good workstations are already free. The Grab algorithm settles for the available number of workstations instead of waiting for the required number to become free. And the Freelist algorithm combines the best features of Grab and Static. All these algorithms are simple and efficient. The Max and the Static attempt to achieve best coordination at the cost of a certain waiting time, while the Grab and the Freelist avoid computing cycles wasting through coordination.

We'd like to develop a coscheduling model which can maximize system utilization, and achieve task parallelism as much as possible. Having done lots of researches on NOWs, we proposes a DynAmic Scalable process miliation & Co scheduling model (DASIC), which can share efficiently and use sufficiently the resources distributed on the network. Besides coscheduling interactive subtasks dynamically, DASIC can also perform allocation and scheduling adjustment to adapt to dynamic system changes.

The main contribution of this paper is the development of an efficient model to utilize available resource dynamically on an NOW, which can guarantee the coordination of parallel applications as well. Section 2 presents the model DASIC. Section 3 describes the design and implementation of DASIC. Section 4 performs simulation and performance study. Section 5 gives the conclusions.

2 The DASIC Model

2.1 Design Issues

Berkeley experience[8] shows that even during the daytime hours, more than 60% of workstations are available 100% of the time. Other experiments found similar results although these statistics are different in detail with different definitions of machine availability. In DASIC, available processors on an NOW constitute a dynamic processor set, called Idle Processor Group (IPG), whose members are probably continuously changing. Here idle processors mean those processors which are available for other users or applications. In an NOW, the number of available processors is not fixed, and the IPG can be expanded or shrunk dynamically.

The goal of DASIC is to coschedule parallel tasks in the IPG, whose size may change from $P_1$ to $P_2$. So those tasks may begin to run in $P_1$ available nodes, and then may scale up or down to run in $P_2$ nodes to adapt to changes of IPG. It is a growing scheduling when $P_2 > P_1$; a shrinking scheduling when $P_2 < P_1$; and a conventional scheduling when $P_2 = P_1$. So IPG management plays an important role in the DASIC model.