Non-Intrusive Information Collection for Load Balancing of Parallel Applications

Stanley Y. Chien, Gun Makinabakan , Akin Ecer, and Hasan U. Akay

Computational Fluid Dynamics Laboratory
Purdue School of Engineering and Technology
Indiana University-Purdue University Indianapolis
723 W. Michigan St., Indianapolis, Indiana 46202 USA
E-mail: schien@iupui.edu

Key words: dynamic load balancing

1. Introduction

Many researchers have studied dynamic load balancing for parallel jobs and various software tools were developed. However, these tools are too specific to certain type of parallel applications. In general, all computation jobs can be divided into three categories. (1) single jobs, (2) loosely coordinated parallel jobs that do not have much communication between parallel processes, and (3) tightly coordinated parallel jobs that periodically communicate among parallel processes. Some job schedulers can provide good job scheduling for single jobs and loosely coordinated parallel jobs but cannot perform well for tightly coordinated parallel jobs. Since many load balancing tools developed for distributed heterogeneous environment are for specific applications (e.g., parallel CFD), they are not suitable for adoption in general computing environment. In this paper, we will describe a scheduling and load balancing tool that can be used for all types of parallel computations.

2. Problem description

A unique property of tightly coordinated parallel job is that the execution time of all parallel processes is determined by the slowest process of the job, since the fast processes must wait information from the slowest process periodically. Therefore, load balancing of tightly coordinated parallel job is to reduce the execution time of the slowest process by assigning it to the fast computer or reducing the load on the computer. Many factors affect the execution time of the parallel processes of a parallel job, such as, the computer speed, the load on the computer, the various problem sizes handled by the parallel processes, the communication among parallel processes, the size of messages among parallel processes, the operating modes of the computation nodes (single user/multiple user, single job/multiple job), etc. Since some factors are not readily available, especially in distributed environment with heterogeneous computation nodes, the existing load balancing approaches require the user to provide some of these factors. This causes a lot burden to the user, thus reducing user's interest of using the load balancing tool.

3. Our past effort

We have successfully developed a scheduler that supports dynamic load balancing of parallel CFD applications on distributed network of heterogeneous computers. These computers can operate in dedicated, batched, or interactive mode. This scheduler also

supports computer fault tolerance for parallel CFD jobs. In order to collect the computation and communication time periodically during the execution of each parallel processes for dynamic load balancing, we require the user to incorporate a timing measurement library into the application code. This library also provides the other needed information for load balancing, such as the size of the data blocks, the connectivity of the data blocks, the handshaking between the application code and the scheduler, etc. However, the requirement of inserting timing library functions into parallel application programs severely limits user's interest of using the load balancing tools.

4. Proposed approach

In this paper, we discuss the possibility of eliminating the requirement of inserting the timing library to the application codes while still supporting scheduling, dynamic load balancing and fault tolerance for any parallel job on networked heterogeneous computers. Our goal is to be able to support the optimal load distribution for all computation jobs. We consider that all information needed for our past scheduling and load balancing approach is still essential. Therefore, we will try to obtain the same information from the outside of the application code. There are several issues need to be studied:

i. Since single jobs, loosely coordinated parallel jobs and tightly coordinated parallel jobs require different scheduling, load balancing and fault tolerance approaches, we need to be able to distinguish these three types of jobs by observing their execution behavior or from the way that the users submit jobs. Single jobs can be recognized easily. One way to distinguish a loosely coordinated parallel job and a tightly coordinated parallel job is to check the elapsed execution time of every processes of the parallel job. If the elapsed execution time for all parallel processes is proximately the same, the job is most likely the tightly coordinated parallel job. Otherwise, the job is a loosely coordinated parallel job.

ii. Without the ability of informing the application code to stop for load balancing and without knowing if a program is stopped or is performing check pointing, the scheduler needs to provide updated balanced load distribution suggestion all time so that whenever the parallel application does check pointing, the parallel application can restart with a balanced load distribution. With the permission from the user, the scheduler can also force the parallel job to do load balancing by force the job to stop and use the fault tolerance features to restart the job.

iii. Without the ability of collecting the work load information directly from the application code, we need to estimate the workload information of each parallel process from the observation of the processes execution. We will use the CPU time of all parallel processes within the same period of elapsed time to determine the relative work load of all parallel processes.

iv. In order to do load balancing, we need to find the relative speeds of computers by executing the same small program on all the computers. The ratio between the CPU time for executing the program of different computers can indicate the relative speed of the

computers. Since the CPU speed of a computer can be different for executing different type of programs, we will use a set of representative small programs to get different speeds of computers. A match method will be used to determine which relative speeds should be used during load balancing.

v. The communication between parallel processes presents a large portion of the execution cost of a parallel job. Therefore, we need to obtain the connectivity information among parallel processes. Since only the experienced users can provide this information, we will try to obtain this information automatically by observing the communication quantity by all parallel processes. For parallel programs that use MPI, we can ask the user to compile the application program with Open-MPI, which had the same functionality as MPI but provide the possibility to measure the number and size of messages send between each pair of parallel processes.

By collecting the load balancing information outside the application code, the developed scheduler and dynamic load balancer will provide an efficient usage of the computer hardware and the convenience to the users.