Simulation of a Parallel Distributed System with Dynamically Allocated Processor Nodes

Andrew Schwartz

June 8th, 2012

1 Introduction

Studies have shown that a majority of parallel jobs do not effectively use available resources to the fullest [4] [15]. One of the main reasons for the underutilization of hardware is that jobs are given a static allocation of processors with which to run their job, while the job itself may require a varying amount of computation power to execute efficiently. Processor allocation policies are conducted two levels: at the lower level, jobs are allocated a set of processors to run on, while at the higher level, jobs decide which instructions run on each of the available processors. A number of studies have been done to try and alleviate the problem of static node allocation through various means, including molding jobs to a dynamic size at runtime [3], and crafting jobs that communicate and synchronize often as to redistribute resources to balance workload across the parallel system [16].

It is not uncommon for a long parallel job to require more computing power at the start of execution than at the end. For instance, the required computation of the LU factorization algorithm tapers off with each successive iteration. One of the most promising techniques for improving the efficiency of a parallel system is have the number of processors allocated to a particular job taper off as a task becomes simpler, as in the previous case of LU factorization. However, in order to efficiently manage this, the job’s resource use must be analyzed over time. Only then can we dynamically alter the number of processors without significant change to execution time.

Simulators including COMPASS [2] and MPI-SIM [14] both use simulation to actively predict the execution time of parallel MPI jobs. This is done using a simulator library that essentially implements most of MPI inside the virtual simulator. With every aspect of the simulation able to be measured, through direct execution of the program inside the simulator, a relatively accurate parallel execution time can be evaluated using a single machine.

However, there is a specific overhead cost of dynamic processor allocation. The distribution costs and data transfers compound to form an overhead operation, not unlike its thread counterpart fork and join. I propose a simulator similar to those above which can simulate the execution of various types of parallel jobs on a single machine utilizing a dynamic processor allocation policy and measure the resulting overhead costs of execution.
2 Background

The goal of efficiently using parallel systems has led to many breakthroughs in processor allocation policies and analysis tools with which to measure the effectiveness of those policies. [4] Since there are two aspects to my research questions, namely the simulator and the job scheduling algorithm, there are two separate research areas which have been studied.

2.1 Simulator

Previous work has been done on developing efficient simulations of parallel systems. [4] There are two main simulators referenced in academia, COMPASS [2] and MPI-SIM [14]. COMPASS, an extension of MPI-SIM, is meant to be a COMponent-based PArallel System Simulator. The goal of these simulators was to accurately simulate large parallel programs to predict the performance of the program on varying architectures. In order to simulate parallel programs, they had to build a way to accurately predict the computation cost, communication overhead, and I/O overhead. They implemented virtual MPI system calls, and created many different components to try and accurately predict the behaviors of applications based on latency, number of nodes, varying caching strategies, and others.

Others have developed similar simulations to accurately determine the runtimes of MPI programs. [14] [2] Some even have created systems with dynamic node allocation, directly in line with what I aim to develop. [15]. I want to consolidate the best principles of all of these simulators and aim to create an efficient simulator to implement the same general features included in all of them, namely simulation of a parallel system and measurement of computation cost, communication overhead, and other overhead costs.

2.2 Job Scheduling

Lots of work has been done on processor allocation policies for distributed systems. [11] [10]. The research in developing a perfect system for processor allocation is ongoing. Research shows that, in the best case scenario, a job scheduler must be developed for a specific purpose. For instance, one can maximize for computational efficiency at the cost of time. Others have more overhead costs due to complex allocation policies, but require specialized jobs to be run on the system. Job schedulers that take into account heuristic values such as expected load over time have been developed in order to try and predict expected load and allocate processors based on those predictions. Some job schedulers can recognize that a specific job can be run on a varying number of processors. For instance, a specific job may be able to be run on any multiple of 2 processors. The scheduler can then allocate the exact number of processors the job requires in order to function best. Other, more complex schedulers, can learn that a specific job ran most efficiently with a specific number of processors and try and manage resources so that for subsequent runs, the job can get that number of processors. The perfect job scheduler must be able to adapt to the different scenarios that arise during the life of a typical distributed system. Specialized job schedulers have been developed for specific needs, but research into creating an efficient general job scheduler is still an ongoing problem.
3 Implementation

The process of determining the overhead cost of a dynamic processor allocation policy in a job scheduler is a two-fold problem. Separately, the job scheduler algorithm and the simulator must be formulated and created. While a job scheduling algorithm can, in theory, be run on both a real parallel cluster and in the simulator, the creation of the algorithm is an independent problem than that of the simulator. Furthermore, a simulator must be developed which can emulate a parallel cluster on a single system. This includes many separate layers which will be further outlined below.

3.1 Job Scheduling Algorithm

The proposed job scheduling algorithm will have the following properties, whose implementation will be described below:

1. Must adapt to various job types, i.e. short/long jobs or computationally expensive/inexpensive jobs.
2. Must adapt to varying cluster sizes.
3. Must optimally allocate processors to jobs, reassigning processors as new jobs are added as to optimize efficiency.

3.1.1 Job Types

In order to maximize the efficiency of the cluster, the job scheduler must take into account the various types of jobs that are submitted to the cluster. For instance, consider the following scenario. Initially, a very computationally expensive job is submitted to the cluster to be run on all nodes in the cluster for a long period of time (on the order of hours or days). This is not atypical of large parallel systems. However, during execution of the initial job, all nodes are being used by the one program. If we submit a short job, which should only take a few seconds to execute, even on just a few nodes, we can end up waiting for the longer job to finish before even starting the short job. This layover time is a problem, as it makes sense that, if we could free up the few processors that the short job needs to execute, we can continue to run shorter programs without interrupting the longer process. In the proposed system, the scheduler would be told a job is a short job, and if applicable, free up some of the nodes for the short job to execute on. After execution of the short job, the now-free processors would be reallocated to the long job to maintain maximum efficiency. Only the maximum number of processors allocated to the process should be specified by the user.

Furthermore, the system should allow for a ranking of importance of jobs. This would be considered another type of job. A lower-priority job would not be able to take processors away from a higher-priority job. In this way, jobs that must be executed right away can be given a higher priority, and be allocated the necessary resources, while lower-priority jobs that can give up the resources can do so as well.
3.1.2 Cluster Sizes

To put it simply, the scheduler must recognize the current state of the system at all times. If the cluster has a certain number of online nodes at the start of execution of a job, and the number changes during execution, it should be able to adapt to this situation and reallocate the nodes as necessary to maintain maximum throughput. Essentially, nodes should be able to log into and out of the system and maintain the execution of all running jobs.

Also, as a given, the system should scale with the cluster size with minimum overhead.

3.1.3 Optimal Allocation

This is the heart of the algorithm, and it works in tandem with the other specifications. When a job is submitted to the cluster, only the maximum number of processors allocated to the job is given by the user. The system itself should use this number simply as a maximum, and instead determine the optimal number of nodes to allocate to the job, given the current state of the system. The scheduler must take into account a variety of factors, including the job-priority, the job-type, the number of currently available processors, and the types of jobs already running on the system. As new jobs are added to the system, the processors should be reallocated so as to maximize throughput of jobs, while also minimizing overhead costs of transferring jobs from one node to another. Work has been done to fold currently running jobs to make way for new jobs through the use of folding, which essentially is taking a job running on $n$ processors, and folding it onto $n/2$ processors by doubling up on the threads on certain nodes. The context switches associated with folding have overhead costs, so it is important to determine if it is worthwhile to fold the jobs in the first place.

3.2 Simulator

Simulators such as COMPASS and MPI-SIM have already done work in simulating the execution of parallel jobs on a single system. I want to build on this idea and create a simulator which can graphically present a parallel system and display the various communication lines and provide an interface for determining overhead and efficiency of the system. The simulator will require multiple layers of implementation, described in the following sections.

3.2.1 Job Scheduler

The job scheduler will be the interface between the jobs and the cluster. In order for a job to be ran on the simulated cluster, it must go through the job scheduler with specific instructions. The job scheduler will take in a requested job, along with the specifications outlined above such as the job-type and priority. After receiving the job, the scheduler will send the job to the various nodes on the virtual cluster.
3.2.2 Virtual Cluster

The virtual cluster is basically a set of virtual nodes which can each run jobs. The virtual cluster will emulate a distributed memory system. In other words, each node will not have access to memory located on other virtual nodes. Nodes must be able to communicate with the job scheduler, so when they go offline, or online, the job scheduler can be notified and adjust its scheduling appropriately. If information wants to be sent from one node to another node, then it must go through the virtual network layer.

3.2.3 Virtual Network Layer

The virtual network layer will simulate the communication channels between nodes on a distributed system. The virtual network layer will have a set of parameters, such as latency, and bandwidth which can be tweaked by the user of the simulator to try and accurately simulate communication over a true network. The virtual network layer will be the communication channel between nodes on the virtual cluster. By trapping the various messages sent across the virtual network layer, we can measure the overhead costs of the job scheduling algorithm.

3.2.4 GUI

The GUI of the simulator should allow parameters of the system to be tweaked by the user, including options for the virtual network layer parameters and the virtual cluster parameters. Also, the system itself should be graphically represented, with communication between nodes shown to the user during execution of a job. Jobs should be able to be submitted by the user running the simulator.

4 Analysis

Analysis of the system will be done by using the virtual network layer of the simulator to keep track of the overhead costs of the job scheduler. The important measurements to be taken by the system are the resulting overhead costs of communication between nodes due to context switches caused by reallocation of nodes to accommodate new states of the system and a measure of efficiency. The efficiency measurement to be compared can be taken by a number of different means, including average wall-time of jobs, wait time before beginning execution of jobs, among others. The simulator will be built with a framework that allows for easy exposure of all relevant data for analysis such as number of communications, time spent communicating, amount of data sent over the virtual network, and anything else that needs to be measured.

It is my intention, time permitting, to implement various job scheduling algorithms that have already been made, and compare the overhead costs to my own algorithm. By comparing the parameters measured using the different algorithms, I aim to research what effects various parameters have on the measurements. Some algorithms, for example, may have less communication between nodes, but more jobs are waiting to be ran. I want to see how each algorithm handles various situations that arise when running a distributed system that handles multiple jobs at once.
5 Concluding Remarks

The development of a proper job scheduling algorithm for efficiently managing parallel systems is an ongoing problem in parallel computing. As more and more of the world’s computations are moving into distributed systems, advancement in the field of job scheduling is a must. Robust, yet flexible, systems are required to manage the various computational needs of the public and of companies. Through research of job scheduling algorithms and the effective simulation of distributed systems, researchers can come closer to developing systems that are more and more fast and efficient.

References


