Centralized and Distributed Job Scheduling System

Simulation at Exascale

Ke Wang

Abstract
Job scheduling systems aim to efficiently manage the distributed computing power of workstations, servers and supercomputers to maximize throughput. As the development of high-speed networking and powerful supercomputers, the scheduling systems should be up to the exascale that is billions of jobs executed by millions of nodes with thousands of cores each, which is however, beyond the state-of-art systems. In this project, our goal is to develop a simulator for exascale distributed job management system, where work stealing is implemented to achieve load balancing. Till now, we have built a centralized simulator using a server to accept jobs submitted by client and dispatch them to nodes. We also built a distributed version, where each node could execute jobs locally, steal jobs from and dispatch jobs to neighbors when needed. Experiment results showed the centralized simulator could handle exascale situation in reasonable amount of time and memory but with low throughput and utility; work stealing worked well for the distributed simulator at small scale, but consumed too much resource to run exascale experiments. We will clearly examining the time and memory consumption of the distributed simulator in order to reduce both, and study the optimal parameters of work stealing to make it more realistic.

Background Information
As a new paradigm, many-task computing (MTC) [1] aims to bridge the gap between HTC [2] and HPC [3]. It emphasizes using large number of computing resources over short periods of time to accomplish many computational tasks. The tasks of MTC may be small or large, uniprocessor or multiprocessor, compute-intensive or data-intensive. In the project, we simplifier that all jobs are independent, per-core and compute-intensive jobs. We need efficient job scheduling systems to improve the job throughput and system utility in order to finish tasks as soon as possible. Job Scheduling Systems could be centralized or distributed. Centralized scheduling system means using a single dispatcher to make decisions such as when and which node to assign jobs, while in distributed systems, each node maintains a job waiting queue, and could steal jobs from or dispatch jobs to others when needed. Work Stealing [4] is an efficient way to achieve load balance, in which the idle processes poll the busy ones to get work to do in order to maximize the throughput and system utility. Several parameters could affect the performance, such as the number of neighbors from which a node could steal jobs, which neighbor to steal jobs and the amount of jobs to steal.

Problem Statement
Though we have already built both the centralized and distributed simulators, the centralized one has the problems of limited throughput and low system utility, and if the server is crashed, the whole system would be down. For the distributed simulator, it consumes too much time and memory to run experiments at even small scale, such as tens of thousands nodes, not to mention the exascale case. What’s more, we found that the optimal number of neighbors a node could have is a quarter of the number of all nodes. This linear relationship is not the real case when the number of all nodes is up to a million. We will pay our attention to the distributed simulator by examining the time and memory
consumption of our program carefully in order to find ways to reduce both. Also, we need to find a more realistic number of neighbors a node could have under the promise of not compromising the performance too much. In addition, we plan to add more complexities of the workloads of the MTC job scheduling systems to make the simulators more robust.

Related Work
A lot of work related to job scheduling systems has been done. Condor [5] was the earliest one developed by the University of Wisconsin. PBS [6] was developed at NASA Ames to address the needs of HPC. LSF [7] was the load-sharing and batch-queuing component of a set of workload-management tools from Platform Computing of Toronto. For work stealing, P. Berenbrink, M. Mitzenmacher and others have proven that given an appropriate work division scheme, the load imbalance under work stealing is bounded making it a stable load balancing algorithm[8, 9]. In [4], the authors proved that random neighbor selection is efficient. In [10], the authors showed that stealing half amount of jobs from the neighbor is optimal instead of just one or two. However, these existing systems are either centralized having the problems of limited throughput, low utility and reliability, or just at the small scale of thousands of cores.

Proposed Solution
In the project, we do discrete event simulations [11] for both the centralized and distributed job scheduling systems in Java. We have already built them.

When analyzing the time and memory consumption of the distributed simulator, we will use the JProfiler [12] software to count the amount of time of each method and the amount of memory of each variable. We will do scalability experiments to carefully examine these on the Fusion machine, which has 48 cores and 256g memory.

After understanding the resource requirements of our program, we may consider to parallelize our program using Java thread by coarsening the granularity.

Evaluation
In the project, we will use the following criteria to evaluate the results.

Time and Memory Consumption: We will collect this information for all the scalability experiments. We expect that it takes several weeks and requires less than 256g memory to run experiments at exscale

Throughput: Number of jobs finished per second. We expect that as the number of nodes double, the throughput would double.

Utility: The ratio of the number of active cores to the number of all the cores at a time. Except the beginning and the end, we expect it as high as 1 at any time.

Coefficient Variance: The coefficient variance [13] of the number of finished jobs of each node. We expect it close to 0 at any time, except the beginning.

Timeline with Weekly Goals

<table>
<thead>
<tr>
<th>Week</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Install JProfiler and learn how to use it</td>
</tr>
<tr>
<td>2</td>
<td>Run scalability experiments using JProfiler to analyze the time and memory consumption</td>
</tr>
<tr>
<td>3</td>
<td>Modify the part of the program consuming too much time and memory</td>
</tr>
<tr>
<td>4</td>
<td>Do the same as week 3</td>
</tr>
<tr>
<td>5</td>
<td>Change the number of neighbors in a wide range to find the realistic and optimal one</td>
</tr>
<tr>
<td>6</td>
<td>Do the same as week 5</td>
</tr>
<tr>
<td>7</td>
<td>Do scalability experiments, collect data, draw graphs</td>
</tr>
<tr>
<td>8</td>
<td>Do the same as week 7</td>
</tr>
<tr>
<td>9</td>
<td>Add more complexities to the workloads</td>
</tr>
</tbody>
</table>

Deliverables
We will have two stable and efficient simulators for both the centralized and distributed job scheduling systems. A mid-term report and a final report will be written to show the progress and results. What’s more, we plan to publish a paper
for this project.

**Conclusion**

We will have a better understanding of the job scheduling systems in MTC, including the type of the workloads, the limitation of the centralized systems and the advantages of the distributed ones. Also, we will have a clear sense of the requirements to build exascale job scheduling systems. What’s more, we could know the scalability of the work stealing approach, which helps examining the feasibility to build a real job scheduling system at exascale.

We will consider the project a success if the results we obtained confirm the criteria listed above. Specifically, it requires reasonable amount of time and memory to run exascale experiments, such as one or two weeks and less than 256g memory. Load balancing will be perfect when applying work stealing at the premise of using the optimal parameters we will find, which means that the increase of the throughput is the same with respect to that of the scale of the systems, the system utility is as high as 1, and the coefficient variance is close to zero.

**References**


