M.S. Project Proposal

Improved Parallel Java Cluster Middleware

Xi He
Rochester Institute of Technology
Department of Computer Science
xi.he@mail.rit.edu

Committee:

Chair: Prof. Alan Kaminsky
Reader: Prof. Hans-Peter Bischof
Observer: Prof. Minseok Kwon

Contents

1 Introduction

In recent years, parallel computing has been widely adopted in domains that need massive computational power, such as graphics, animation, data mining and informatics. While the parallel programs are in general written in C, as Java has become one of the most popular languages in the IT industry with its "write once, run anywhere" feature and powerful support from open source organizations and big IT vendors, a Java base parallel computing framework is regarded as necessary.

Parallel Java (PJ) is an API and middleware for parallel programming in 100% Java developed by Professor Alan Kaminsky [1]. Figure 1 shows the architecture of PJ running on a cluster parallel computer with one frontend node and multiple backend nodes connected by a high-speed network. A Job Scheduler daemon and multiple Job Frontend processes run on the frontend nodes. The Job Scheduler keeps track of each backend node's status and maintains a queue of PJ jobs; The Job Frontend connects to the Job Scheduler, and spawns a Job Backend process on the backend node. Then the Job Backend communicates with Job Frontend, obtaining the programs class files and command line arguments, and calling the static main() method of the main program class. The Job Frontend relays the jobs standard input, standard output, and standard error streams between the Job Backend and the users terminal. A web interface in PJ (See Figure 2) displays the status of the cluster parallel computer.

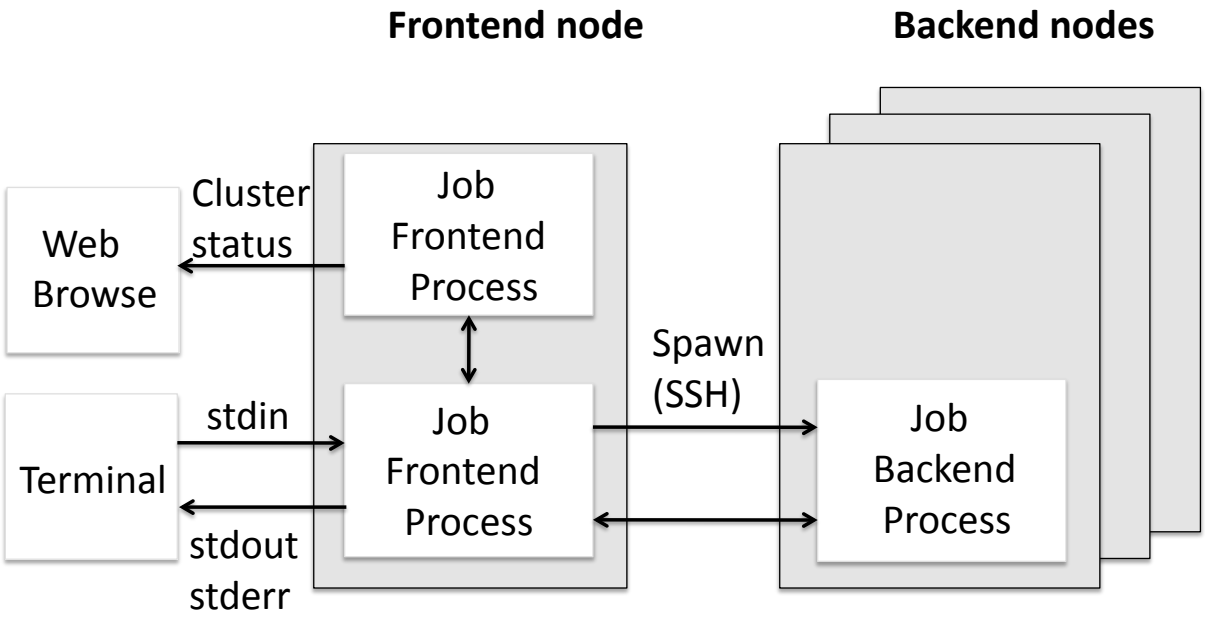

Figure 1: Parallel Java Architecture

The goal of our master project is two-fold. One is to design and implement a more efficient job launching scheme that is faster and doesnt require SSH. Another goal is to enable PJ users to perform more tasks via PJ's web interface, such as job submission or job cancelation.

2 Related Work

Besides PJ, there exist different flavors of MPI implementations, such as MPICH2 [2], OpenMPI [3] and mpiJava [4]. As to the job launching scheme, we plan to have a complete comparison of these implementation with PJ.

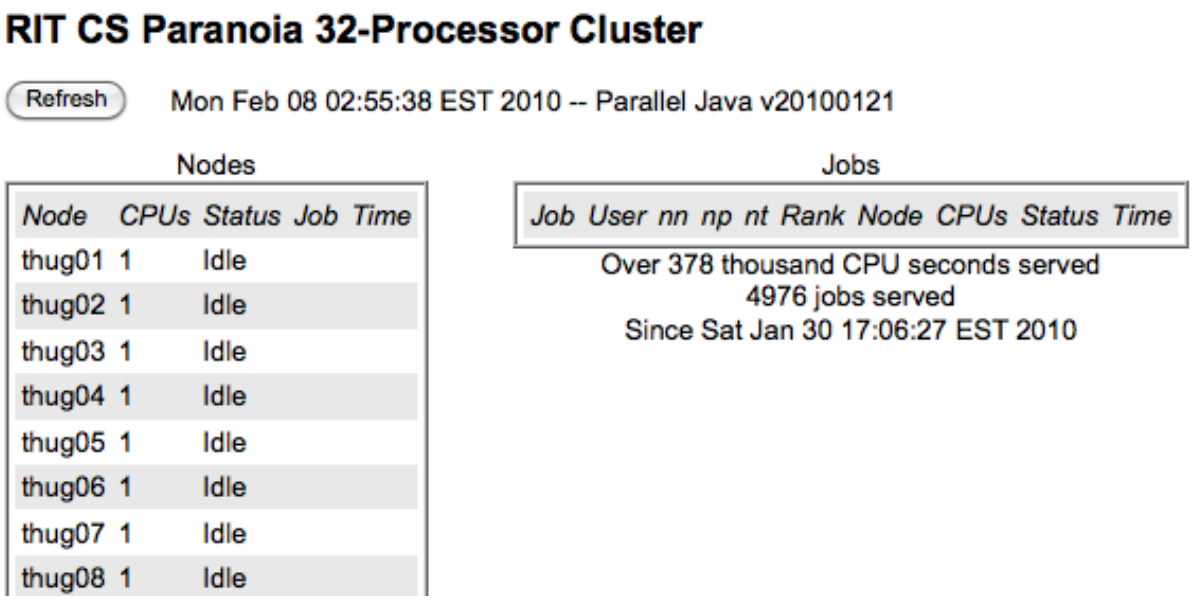

Figure 2: Parallel Java Web Interface

3 Design

In this section, we present a complete analysis of the two tasks in our project and propose our solutions to the problems.

Our first task is to design and implement a more efficient PJ job launching scheme. The working of current PJ job launching scheme is as follows:

- During system installation stage, system administrator launches the Job Scheduler daemon on the frontend node.
- When an end user submits his parallel program to the cluster, a Job Frontend process on the frontend node is started.
- The Job Frontend process connects to the Job Scheduler daemon and requests for available computing resources.
- After obtaining enough resources from the Job Scheduler, the Job Frontend will use SSH to start a Job Backend processes on each assigned backend nodes. Then these Job Backend processes start to launch the job.
- When the job finishes, the results will be relayed to the user terminal via the Job Frontend process.

A typical SSH conversation involves authentication, keys exchange and encryption, which create many processes and occupy plenty of resources. Current implementation of PJ spawns a separate SSH session for each backend processor to start each backend process and authenticate each backend process into the users account. As a result, PJ job launching scheme is somewhat inefficient. To address this problem, the new job launching scheme would adopt authenticated message passing to assign the jobs to the backend nodes instead of encrypted message passing which is more time-consuming.

Figure 3 shows the new job launching scheme. When the end user first submits his job to the frontend node, a Job Frontend process will be started and the Job Frontend process will then use SSh to launch a Job Launcher daemon running on behalf of the end user on each assigned backend node. The Job Launcher daemon spawns a Job Backend process on the backend node which will be responsible for running the user's job. Next time when the end users is assigned the same backend node to run his job, instead of using SSH, he can only send the MAC message to the Job Launcher daemon on the backend node, and request the Job Launcher daemon to spawn a Job Backend process to run his job.

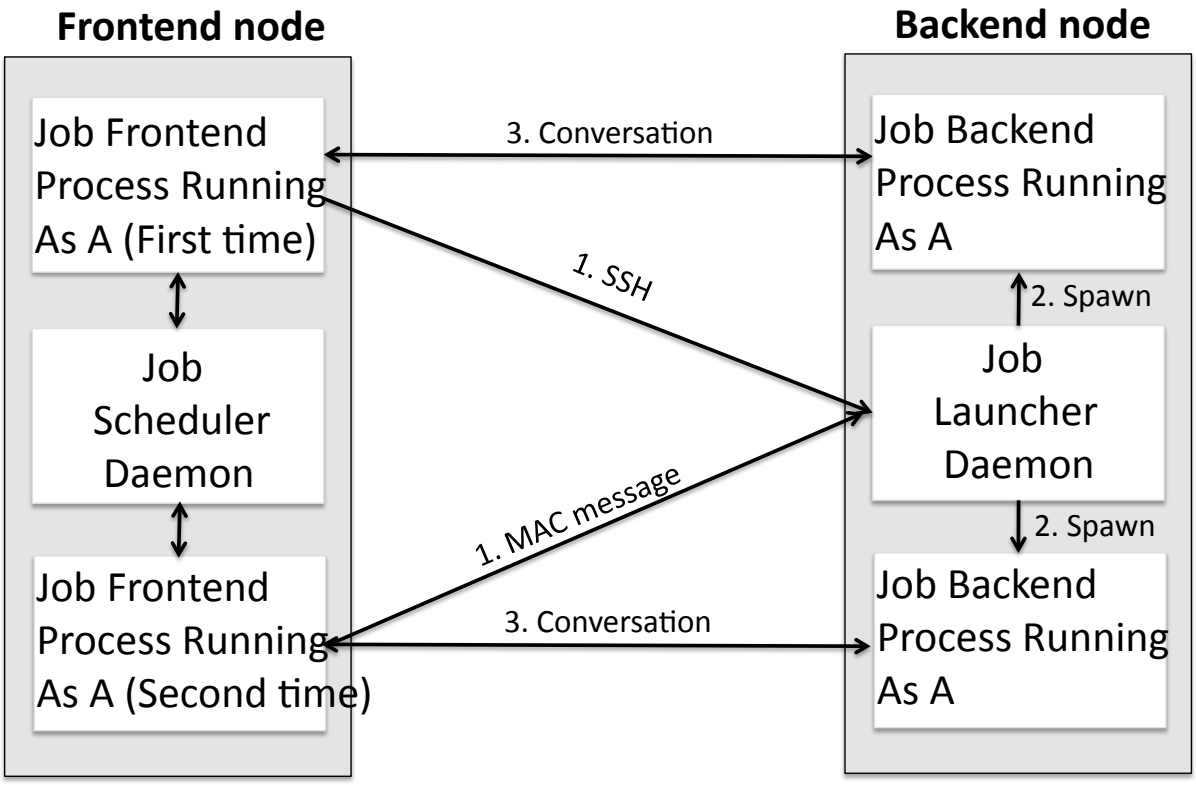

Figure 3: New Job Launching Scheme

We do not want to develop the new job launching scheme from the scratch. Instead we can modify the existing job launching scheme to achieve a more efficient job launching scheme. Here is what we plan to do:

- Add a Job Launcher class. This class is responsible for MAC base authentication and spawning job backend process on behave of the end user on the backend nodes.
- Modify JobScheduler class.
- Modify JobFrontend class. The Job Front process request job execution by sending MAC message instead of using SSH.

The second task of our project is to design and implement a new web interface with more functionalities. Right now PJs web interface only displays the PJ job queue status. The new web interface is designed to present different views and grant different privilege to different users.

As to anonymous users, the web interface (Figure 4) shows the status of the nodes and the job queue in the cluster.

As to common users, the web interface (Figure 5) is similar to that of anonymous users except that the common users can submit jobs to the cluster and run the jobs through the web interface. The common users can also cancel the jobs he submitted.

As to system administrators, he can cancel any jobs submitted. He can also change the priority of suspending jobs. (Figure 6)

Figure 7 shows the web interface architecture. A secure connection is developed between the web browse and Job Scheduler daemon on frontend node. When a web interface user submits a job to the PJ cluster or cancel a job running or suspending in the PJ cluster, the web browse sends the end user's request and his credential to the Job Scheduler daemon. The Job Scheduler daemon will then try to log in to the user account with the provided credential.

RIT CS Paranoia 32-Processor Cluster

Mon Feb 08 16:39:49 EST 2010 -- Parallel Java v20100121

Refresh

Nodes

Node	CPUs	Status	Job	T
thug01	1	Idle		
thug02	1	Running	4979	10
thug03	1	Running	4979	10
thug04	1	Running	4979	10
thug05	1	Running	4979	10
thug06	1	Idle		
thug07	1	Running	4980	20
thug08	1	Running	4980	20
thug09	1	Idle		

Jobs

Job	User	Status	Nodes	CPUs	T
4979	xxh2229	Running	thug02	1	10
			thug03	1	10
			thug04	1	10
			thug05	1	10
4980	jxd2352	Running	thug07	1	20
			thug08	1	20

Figure 4: PJ's web interface for anonymous users

Welcome, xxh2229

RIT CS Paranoia 32-Processor Cluster

Mon Feb 08 16:39:49 EST 2010 -- Parallel Java v20100121

Refresh Logout

Nodes

Node	CPUs	Status	Job	T
thug01	1	Idle		
thug02	1	Running	4979	10
thug03	1	Running	4979	10
thug04	1	Running	4979	10
thug05	1	Running	4979	10
thug06	1	Idle		
thug07	1	Running	4980	20
thug08	1	Running	4980	20
thug09	1	Idle		

Jobs

	Job	User	Status	Nodes	CPUs	T
X	4979	xxh2229	Running	thug02	1	10
				thug03	1	10
				thug04	1	10
				thug05	1	10
	4980	jxd2352	Running	thug07	1	20
				thug08	1	20

upload Required CPUs

Args Run

Figure 5: PJ's web interface for common users

If successful, the Job Scheduler Daemon can spawn a Job Frontend Process to accept the user's job submission or

Welcome, administrator

RIT CS Paranoia 32-Processor Cluster

Refresh Logout

Mon Feb 08 16:39:49 EST 2010 -- Parallel Java v20100121

Nodes

Node	CPUs	Status	Job	T
thug01	1	Idle		
thug02	1	Running	4979	10
thug03	1	Running	4979	10
thug04	1	Running	4979	10
thug05	1	Running	4979	10
thug06	1	Idle		
thug07	1	Running	4980	20
thug08	1	Running	4980	20
thug09	1	Idle		

Jobs

	Job	User	Status	Nodes	CPUs	T
X	4979	xxh2229	Running	thug02	1	10
				thug03	1	10
				thug04	1	10
				thug05	1	10
X	4980	jxd2352	Running	thug07	1	20
				thug08	1	20
Dn	4981	lxh3218	Waiting		5	
UP	4982	wyx3424	Waiting		8	

Figure 6: PJ's web interface for system administrators

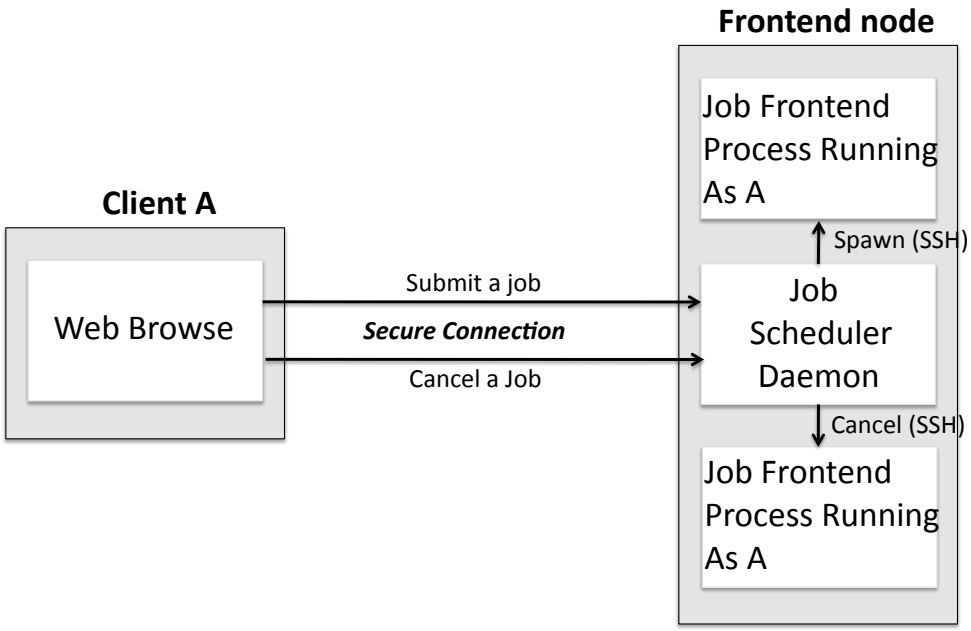

Figure 7: PJ's web interface architecture

cancel the running Job Frontend Process. Otherwise it will reply an error information to the web interface user.

4 Test

The test of the job launching scheme is to compare the time the new job launching scheme of PJ, the old job launching scheme of PJ and the job launching scheme of a few MPI implementation such as MPICH2 or OpenMPI take to start a job. The test program submits 1000 jobs to different job launching schemes, respectively. Each job requires all processors and its task is just to print out the current date of the backend nodes to a log file. By analyzing the log file, we can compare the efficiency of different job launching scheme.

We also want to compare the efficiency of submitting a job via PJ's web interface with that of the old and new job launching schemes. We submit 1000 jobs from the web interface with each job requiring all processors and then calculate the time for the jobs to start. By comparing this result with the above test results, we can have an analysis of different job submission mechanisms.

5 Deliverables

Upon completion of this project the following deliverables will be presented:

- Final report containing
  - Detailed job launching scheme and web interface design
  - Summary of implementation
  - Performance study
  - Web interface user's manual
  - Future work
- Source code of job launching scheme and web interface.
- Test programs

6 Project Timeline

The following is a tentative schedule for the completion of major phases of this project:

- Proposal Completed 02/19/10
- Background Research 02/26/10
- Implementation of job launching scheme 03/12/10
- Implementation of web interface 03/26/10
- Testing & Benchmarking 04/02/10
- Project Final Paper 05/07/2010
- Project Defense 05/21/2010

References

[1] A. Kaminsky, "Parallel Java: A unified API for shared memory and cluster parallel programming in 100% Java," in *21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*. Citeseer, 2007.

[2] "MPICH2," Website, 2009. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpich2/

[3] "Open MPI," Website, 2010. [Online]. Available: http://www.open-mpi.org/

[4] "mpiJava," Website, 2007. [Online]. Available: http://www.hpjava.org/mpiJava.html