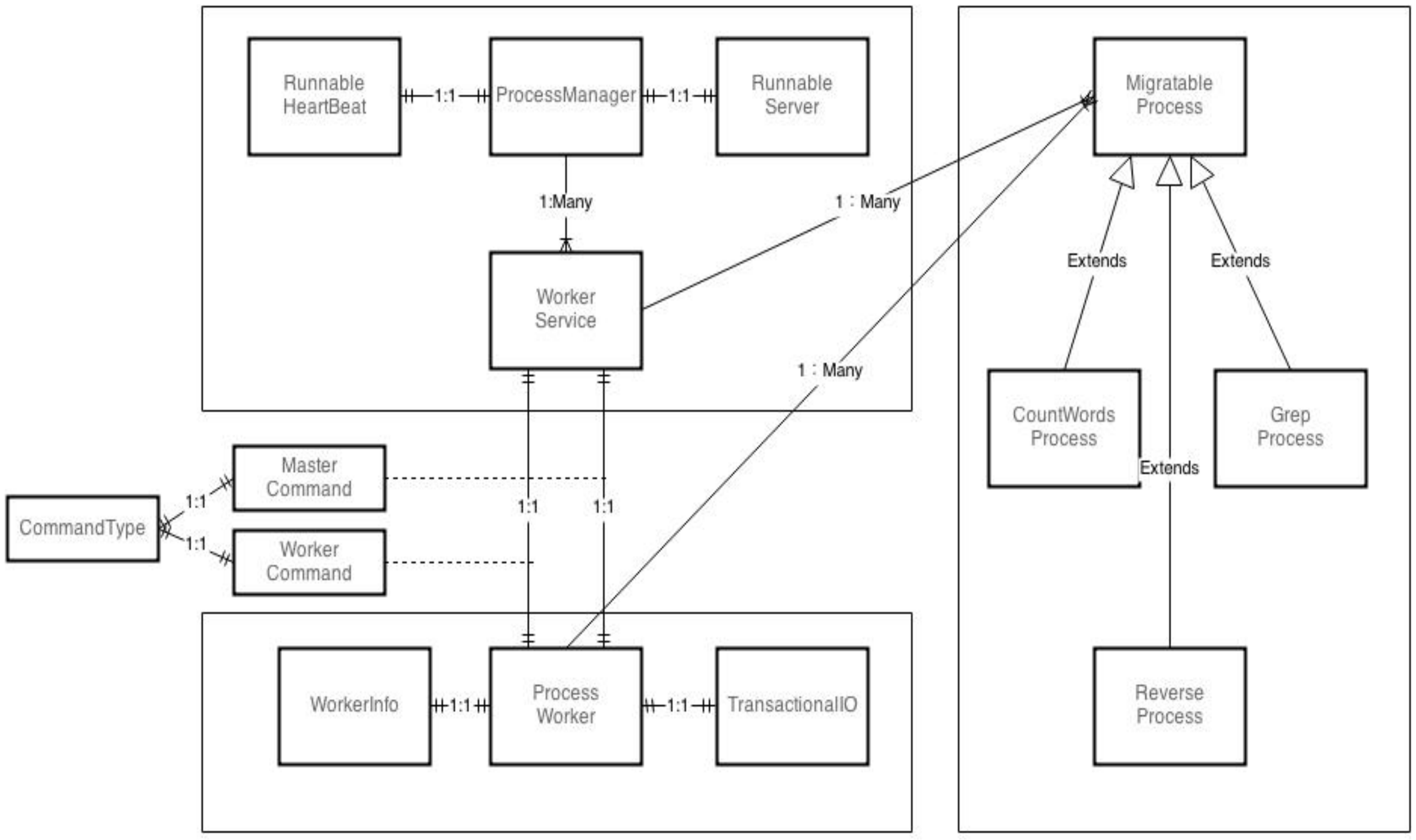
15640 Project1 Report

Name1: Xincheng Liu
AndrewID: xinchenl
Name2: Hao Ge
AndrewID: hge

Part I: Design

For the architecture, we choose a classic master slave model: the master is responsible for managing all processes and slave is responsible for execute all processes. The architecture is showed in the follow:



Here is a basic introduction of each class and some design consideration behind it.

1. ProcessManager

Open the console and start heartbeat and server thread. It will send commands such as migrate, start and kill to manage the worker. It will keep information of process and worker.

2. RunnableHeartbeat
Send information to ProcessWorker every 10 seconds. If the worker fail or the process fail or the process is completed, the ProcessManager will be notified.

3. RunnableServer
Handle all connecting socket and start service thread for each connection

4. WorkerService
Handle all connection with one worker, it is also responsible for sending migrate process to target worker.

5. ProcessWorker
Executing process and return all information it has every heartbeat.

6. TransactionalIO
It uses the randomAccessFile to move position in file. It also keeps a variable – offset to record the writing position in the file. These two designs ensure the in-progress moving without disruption and wastage.

7. MigratableProcess
Here is a tradeoff. When we implement concrete processes such as GrepProcess or ReverseProcess, we find that if the process doesn't flush and close the IO stream, there will be some problem in migration. One solution is to wrap all file operation including file closing in parent class: MigratableProcess, but it's complicated and a little bit wired because the sub-class cannot even use BufferedReader to wrap a fileIO, so we decide to let sub-class handle fileIO.

Part II: Features

1. Fault Tolerant
Our implementation is fault – tolerant to some situations: Failure on worker, failure on starting process and failure on executing process. We catch different exceptions and send out the fail information. The detail test is discussed in Test Case.

2. Real – Time information update
The master server should know whether a worker is failed, whether a process is running and most importantly, it should know whether a process is completed. In our implementation, all these information will be displayed with "ps" and "ls"

command in master server. We use the StatusType class to show different status.

3. Low workload on server side

The server is only responsible for sending command and receiving command. All the thread is started and resumed on slave side. We think this is important to improve the performance.

Part III: Deployment

Require: JRE Java 1.7, Linux, AFS file system

1. Copy source code to each machine
2. "cd" to the **MigratableProcess/bin** directory
3. For master machine:
   Type in: **java edu.cmu.ds15640.core.ProcessManager <port number>** in terminal
   Example: **java edu.cmu.ds15640.core.ProcessManager 8888**
4. For worker machine:
   Type in: **java edu.cmu.ds15640.core.ProcessWorker <master IP address> <port number>**
   Example: **java edu.cmu.ds15640.core.ProcessWorker 128.2.13.134 8888**

5. When the worker machine joins the master machine there will be notice in the master machine: "One worker joins"

Part IV: Test Case

1. Command for ProcessManager:
   a) **ls**: list all workers, each worker has a unique id. The worker id can be found in this command
   b) **ps**: list all processes, each process has a unique id. The process id can be found in this command
   c) **help**: list all command information
   d) **kill <process id>**: stop and remove the process
   e) **start <worker id> <process name> <process argument 1> <process argument 2>…** : start a process in a worker machine. The process argument needs to match the requirement of the process you want to run
   f) **migrate <process id> <source worker id > <target worker id>** : migrate a process from one worker to another. Source worker is the one originally running the process. Target worker is the one you want to move the process to.
2. Example:

a) GrepProcess
Usage: GrepProcess <queryString> <inputFile> <outputFile>

- **Start:** To assign this process to a worker, please use **start** command (make sure master and worker connected successfully)

  Type in: **start <worker id> edu.cmu.ds15640.process.GrepProcess <queryString> <inputFile> <outputFile>**

  Example: **start 0 edu.cmu.ds15640.process.GrepProcess THE /afs/andrew.cmu.edu/usr15/hge/private/15640/MigratableProcess/inputfile.txt /afs/andrew.cmu.edu/usr15/hge/private/15640/MigratableProcess/outputfile.txt**

- **Migrate:** To migrate a process from one worker to another, please use **migrate** command (make sure use **ps** command to get the process id and worker id first)

  Type in: **migrate <process id> <source worker id> <target worker id>**

  Example: **migrate 10000 0 1**

- **Kill:** To kill a process, please use kill command (make sure use **ps** command to get the process id)

  Type in: **kill <process id>**

  Example: **kill 10000**

b) CountWordsProcess
Usage: CountWordsProcess <inputFile> <outputFile>

For **start** command below is the example

Example: **start 0 edu.cmu.ds15640.process.CountWordsProcess /afs/andrew.cmu.edu/usr15/hge/private/15640/MigratableProcess/inputfile.txt /afs/andrew.cmu.edu/usr15/hge/private/15640/MigratableProcess**

/outputfile1.txt

For **migrate** and **kill** command they are same as the GrepProcess one

c) ReverseProcess
Usage: ReverseProcess <inputFile> <outputFile>

For start command below is the example

Example: **start 0 edu.cmu.ds15640.process.ReverseProcess /afs/andrew.cmu.edu/usr15/hge/private/15640/MigratableProcess /inputfile.txt /afs/andrew.cmu.edu/usr15/hge/private/15640/MigratableProcess /outputfile.txt**

For **migrate** and **kill** command they are same as the GrepProcess one

3. Suggested Test Procedure

a) Basic test
Launch one master machine and two worker machines. Type ls command to list all the workers. Start a process in worker 0 and migrate to work 1. Type ps to get process status. Migrate process from worker 0 to worker 1 and check the status again. Kill the process or wait for a work finishing the job.

b) Fault Tolerance
   i. Quit any worker machine at any time
   ii. Argument check in both master and worker side
   iii. When master machine lost connection worker will finish the process and quit
   iv. If worker fail in any time, master will get fail information at any time by heartbeating.