15-640 Project1 Report

Name 1: Rui Zhang
AndrewID 1: ruiz1
Name 2: Jing Gao
AndrewID 2: jinggao

Part I: Design and Features

The program demonstrates the migrations of migratable processes between nodes of a distributed system.

There will be one master node taking care of the management work -- maintaining a distributed network, accepting jobs from user input, designating processes to certain nodes and migrating them between the nodes, periodically balancing loads by migrating processes, and periodically checking unexpected failures and the status of each slave node. **At the same time, the master node will be able to share workload from the slave nodes, performing slave functionality whenever average workload of slaves exceeds a threshold. This also ensures that if there’s no slaves running in the system, the system can also operate properly.**

We also support manual operations of migrating, suspending, resuming and terminating processes. This feature enables users to make processes running on whatever state and whichever slave they want. And along with the feature of printing out processes and slaves information, one can check the correctness of the system’s operation.

Slave nodes will connect to the master node via socket on the provided host and port. A slave acts according to the messages sent by the master node, running designed migratable processes, and migrating, suspending, resuming, or terminating a certain process whenever required by the master. A slave node will perform management functionality only locally. Similar to master node, local management in a slave node includes keeping an updated list of local threads and corresponding processes, and periodically checking and reaping local failures.

Messages are useful in master-slave communications. Encapsulated in a message will be a command/response string describing the job, associated slave/process info, and the serialized migratable process that is specific to the job. Transmission of not only commands/status, but also serialized processes themselves, is considered a feature of this program.

Two migratable processes are designed as examples. They implement functionalities that will facilitate migration, such as suspend, resume, and terminate. Also included in the migratable interface are transactional I/O, which will enable migrating processes with file I/O to continue file I/O on new nodes and skip already read or write bytes. **Caching of file connections is implemented via utilizing transient key words and migrated flag. Transient key words will ensure the inputStream and RandomAccessFile objects inside the transaction I/O objects won't serialized and sent while the migrated flag will ensure that the inputStream and RandomAccessFile won't open and close repeatedly if the flag is set false which means the process has not been migrated yet. By doing this, it will enhance performance of file I/O.**

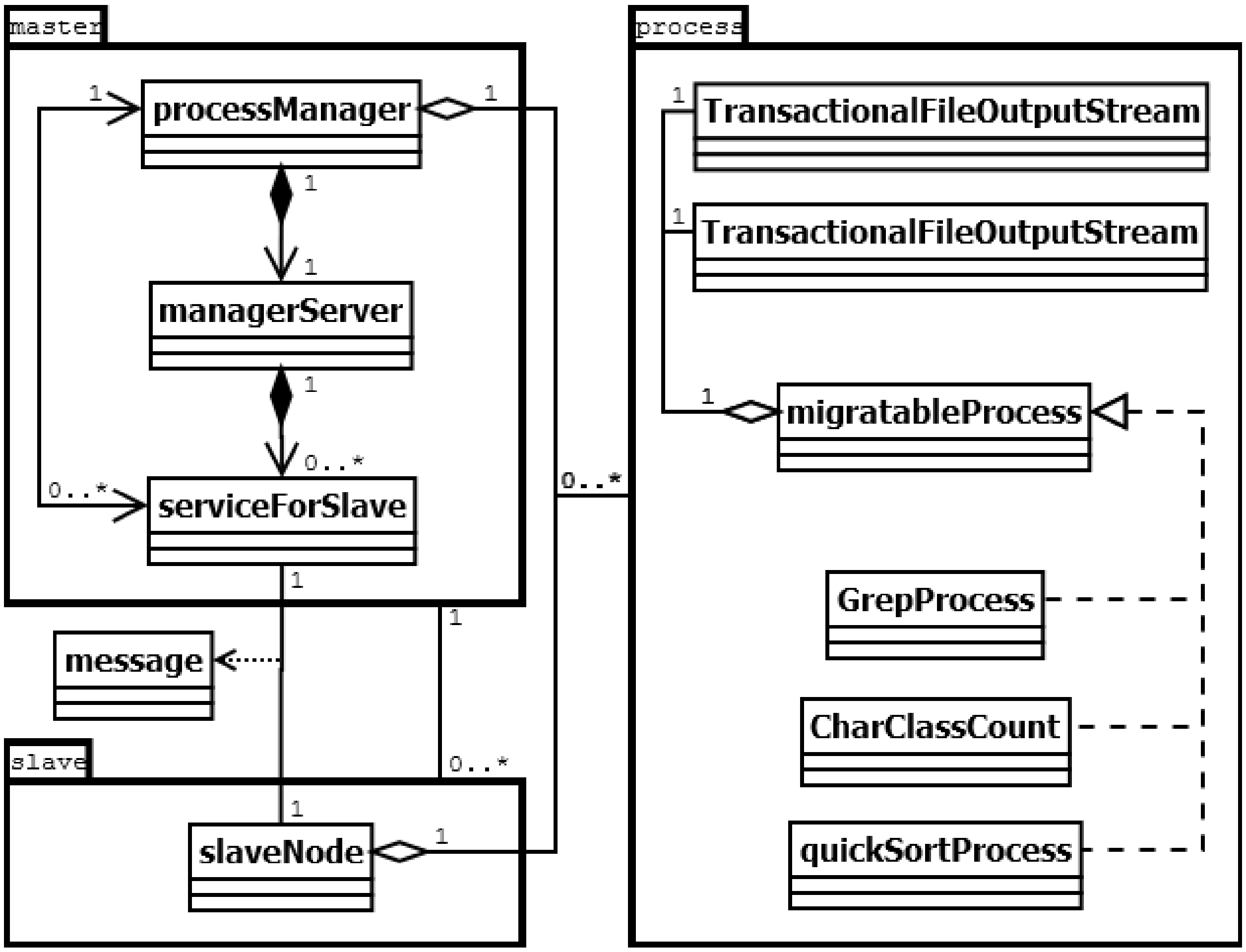

Figure 1. Class Diagram

Class diagram is given in Figure 1. Detailed descriptions of the structure and some features are as follows.

A. Master class group:

A1. processManager. It includes main() on the master node. In this class, hashmaps such as pidToProcess, pidToSlave, sidToSlave, and sidToLoad, as well as a linkedList of suspended processes are maintained. These info facilitate the management of process migrations. Main thread continuously scan for user input, and deal with user requirements by arranging local processes or by forwarding requirements to specified slave nodes. The main thread also deals with responses from slave nodes and update info in the hashmaps and linkedlists accordingly. **A reapTimer which reaps local failures every 5 seconds and a balanceTimer which balances workload every 7 seconds are alive and working during the lifetime of the master node.** Initial load for master is assigned 10, indicating extra management work as master. Workload increments by 1 as any process begins to run on a node.

A2. managerServer. It is a long living thread on the master node, created by processManager in early stage. It keeps accepting connections from slave nodes.

A3. seviceForSlave. Upon receiving a connection from a slave node, managerServer creates a serviceForSlave thread. A serviceForSlave provides socket I/O streams to and from the slave, keeps reading its responses and forwarding them to processManager.

B. Slave class group:

B1. slaveNode. This class takes care of all works on a slave node. The main thread starts with socket connection to the master node, then it keeps reading and dealing with command from the master node. It maintains hashmaps pidToProcesses, pidToThread, and linkedList suspendedProcesses. A reapTimer reaps local failures every 5 seconds is alive and working during the lifetime of the slave node.

C. message. A message between master and slaves has four fields: sid, global slave id assigned by processManager, starting 0; pid, global process id assigned by processManager, starting 0; serialized migratable process; command string. There are 6 types of command string: “C” for connect; “R” for run; “M” for migrate; “S” for suspend; “E” for resume; “T” for terminate. “C” is the only type of message that slaves take the initiative to send to the master; all of the other five types of message are initialized by the master and only responded by the slaves.

D. Process class group:

D1. TransactionalFileInputStream and TransactionalFileOutputStream. Transactional I/O classes extend Java I/O stream classes and implement Serializable interface. To facilitate migration, the transactional I/O record current position in the files to read/write, and seek to the position and resume reading/writing after migration. To avoid consistently opening/closing files, caching of file connections is implemented; when the migrated flag is not set, file connections will be cached.

D2. migratableProcess. This is an interface that all of the migratable processes should comply with. Included methods are suspend(), resume(), terminate(), toString(), and getInput(), getOutput() which return transactional I/O.

D3. GrepProcess, CharClassCount, and quickSortProcess. These are three example migratable processes. GrepProcess is provided along with project requirement and will grep each line of an input file to an output file. CharClassCount counts the appearances of consonants, vowels, digits, spaces and punctuations in each line from an input file and print the stats to an output file. QuickSortProcess implements quick sort algorithm on characters of all lines from an input file and print the sorted characters to an output file.

Please refer to Figure 2 (sequence diagram of process migration) for a generalized workflow of the program.

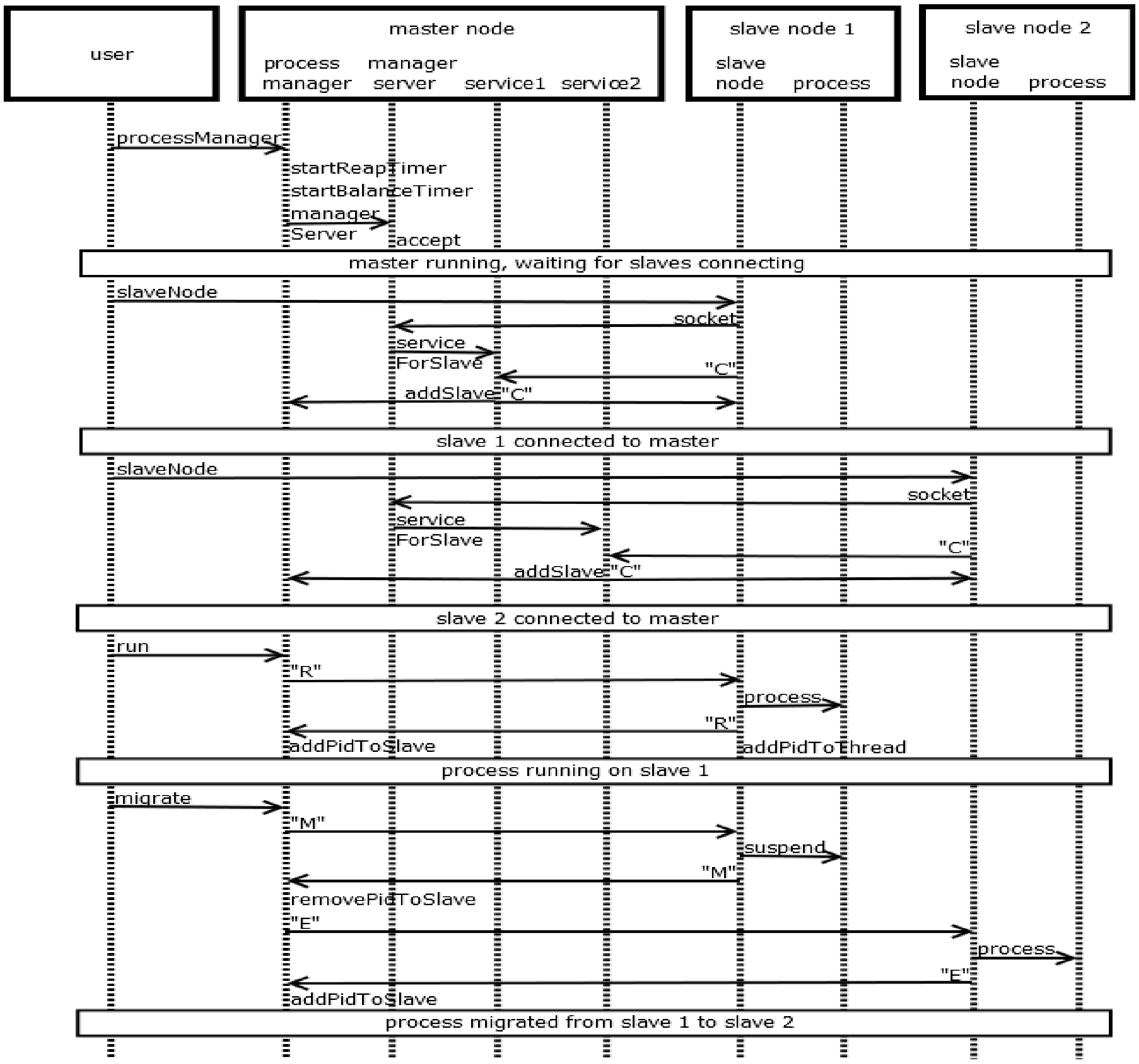

Figure 2. Sequence Diagram

Part II: Implemented vs Umimplemented/Bugs

A. Implemented:

We have implemented all the features mentioned in the document and implement some other features.

A1. MigratableProcess interface

In our framework, we have not only implemented the basic toString()[To print out the class name of the current process], suspend()[To suspend the current process which make it into safe state] methods, but also provide methods like **resume()**[To resume a suspended process] and **terminate()**[To terminate a process].

In addition, all of our migratable processes take an array of strings as their only argument. All the I/O operations are done through transactional I/O described below.

A2. Transactional I/O

We have implemented transactional I/O which extends java.io.InputStream and java.io.OutputStream respectively and both implements the java.io.serializable interface.

To continue perform I/O operations, these two classes have a position variable to record the read or written bytes. When having transferred to another node, it saves repetitive reads or writes after seeking to the specified position.

To further boost performance, we **cache connections by setting a migrated flag** as the document states. The flag is set when process is migrated and reset when the process resumes on another node and begins to read or write. Unless the flag is set, the file won't close. This saves the file opening cost every time reading or wring a file on the same node.

A3. ProcessManager

We have implemented a processManager to lauch, suspend, resume, migrate and terminate processes. We have chosen to periodically check the whether the processes are dead through a timer every five seconds.

To distinguish processes from each other, we assign every process a unique process id.

The manager is also responsible for input command parsing, sending messages to slaves, doing corresponding jobs when receiving messages from slaves and maintaining processes' and slaves' information.

A4. Examples

We have implemented two kinds of migratable processes: quicksortProcess and CharClassCount. The first one is to read all characters from input file, sort them with quick sort algorithm and then output to the output file. The latter one is to count the appearances of consonants, vowels, digits, spaces and punctuations in each line from an input file and print the stats to an output file

Part III: How to Build, Deploy, Run

1. copy program to each node
2. cd into migrationProcess directory
3. make clean
4. make
5. on master node, java processManager <port>
6. on slave node, java slaveNode <master host name> <master port>

Part IV: Dependencies and Software/System Requirements

Operating System: Linux
JRE: Java 1.7.0

Part V: Test Framework with Two Examples

1. Run the master node with inputs below:
   a. quit: quit the program
   b. ps <option>:
      "-sp": print suspended processes' information with the order "ProcessId, ProcessName, Process state, slave ID, slave address".
      "-p": print all processes' information with the order "ProcessId, ProcessName, Process state, slave ID, slave address".
      "-sl": print all slaves' load information with the order "SlaveID, slave address, slave load"
      "-s <slave id>": print information of the slave with specified slave id in the order "Slave ID, Slave Address, Slave load and then information of all the processes running on this slave".
   c. migratable process's name and its arguments : running an instance of that process
   d. resume <process id>: resume the suspended process with specified process id
   e. suspend <process id>: suspend the process with specified process id
   f. terminate <process id>: terminate the process with specified process id
   g. migrate <process id> <slave id>: migrate the process with specified process id to slave node with specified slave id

2. Run the two examples:
   Input the example classes' name and its necessary arguments on the master node's program.

   CharClassCount: Input "CharClassCount <input file name> <output file name>"
   QuickSortProcess: Input "quickSortProcess <input file name> <output file name>"