**Report**

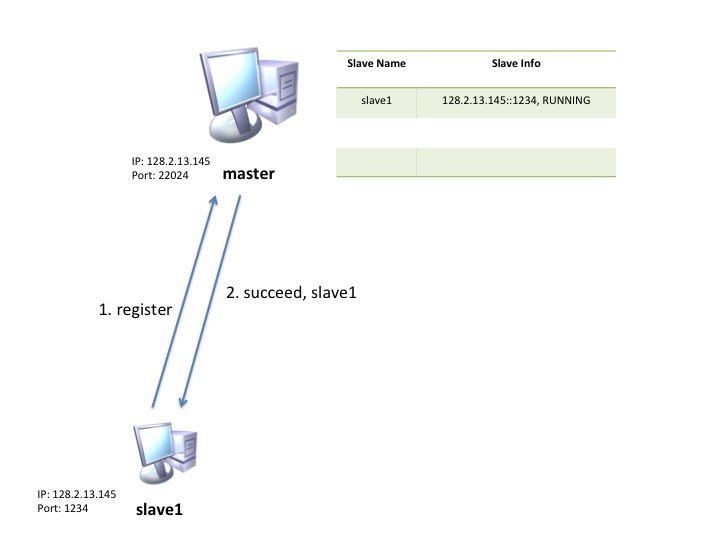
1. **Design**

The distributed system consists of one master node and several slave nodes. Node refers to the process running on a host in this report. Master node acts as namespace manager, which is responsible for tracking slave nodes, assigning names for each slave node, and handling slaves information query. The slave node takes charge of registering/deregistering itself on master node, executing commands users type in from stdin, migrating processes and accepting migration of processes from other slave nodes. For simplicity, this system resides on a distributed file system.

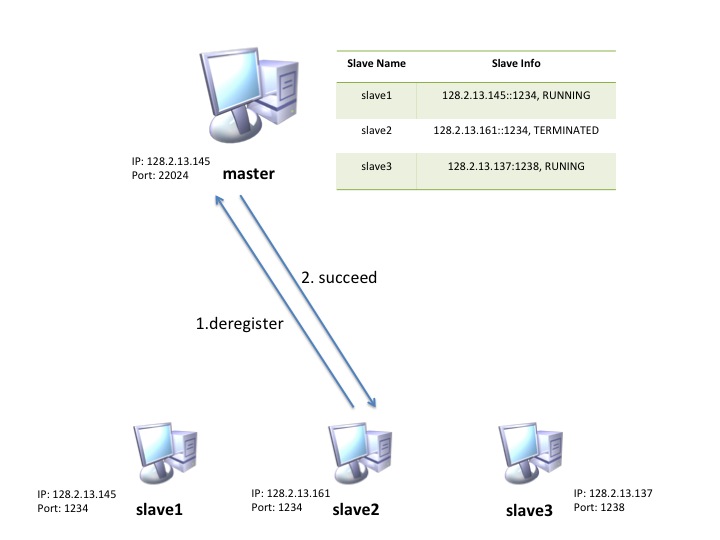
* 1. **Master Node**

When master node starts, master node will listen at a “well-known” port that all slave nodes are aware of before hand. This listening port will accept and handle kinds of request from slave:

* Register: Slave should be able to register on master node. As shown in (Figure 1, master keeps a table that keeps track of each slave node’s information. And when a slave starts, it sends the registration request to master node, master node should either register it on the table or deny its request. Each slave is globally recognized by the name that master assigns to it. And each slave is recognized by the tuple of <host-ip, listen-port> at the master node in order to avoid multiple processes of same host listening at same port..
* Deregister: When slave node closes, it is expected to notify the master node. The slave node will send deregister request to master node as shown in (Figure 2. And the master will change the corresponding slave information to indicate such slave is terminated.
* Query: The master also handles the slave node query. Each slave node may query either all the available slaves information for sake of usability or specific slave information in order to migrate the process to destination slave node.



(Figure slave1 registers on master)



(Figure slave2 deregisters on master node)

* 1. **Slave Node**

Slave node will run the process and handle migration issues of each process. It first initializes and then runs into work mode. When it receives a quit command, it runs into clean-up mode.

* Initialization: When slave node starts, it firstly registers itself on the master node with port it is listening to for receiving MigratableProcess object. If the port is used by other processes on the same host, it shuts down and users need to restart it again with another unused port. If the IP-port tuple shows used and terminated before, the master just refreshes the old slave name and returns the old slave to the slave node.
* Work Mode: When slave node successfully registered on master node, it starts a peer thread (we call it MigrationHandler) which waits for migration request from other nodes. The main thread handles the users command from users via stdin.

The slave can launch any MigratableProcess that conforms to my interface with a restriction that it should be available before compiling time.

When slave starts to migrate, it firstly retrieves the destination slave node information, such as inet address MigrationHandler listening port and its availability. If the slave is not available, slave should not suspend the process and migrate it. Otherwise, it will suspend the process, wait for all instance variables are correctly saved and send the process to the destination slave node. The destination slave node receives the migratable process via MigrationHandler and resumes the process which was suspended by previous slave node.

* Clean-Up Mode: When the slaves starts to close, it first terminates MigrationHandler thread, so that no further migration process will be accepted by the slave node. Then it terminates all the migratable processes that are running on the node. Finally, slaves deregister itself on master node so that other slave nodes are not able to migrate a process to it.
  1. **Migratable Process**

Mitratable Process refers to objects that implements MigratableProcess, Runnable and Serializable. As suggested in the writeup, MigratableProcess constructor takes the String[] as arguments in order to provide the flexibility to run other classes that implements MigratableProcess.

The process which is supposed to be migratable should follow the paradigm of GrepProcess.java. The run() method functions and work on the task. Inside of run() method of MigratableProcess, there is a loop which takes a shared variable called suspending as its condition to allow slave node suspends each process. Due to the restriction of this paradigm, all the progress state of each process are required to be stored in instance variables and all needs to be Serializable.

The suspend() method is for purpose of suspending each process. Once call the suspend() method, the suspend() method should block to wait for the process to save all the states of the process. After the process saves all the information, it notifies the suspend() method. In case of blocking on a process which is not running, a finished variable is added to make sure the call on suspend() won’t be blocked forever

1. **Bugs And Things Unimplemented**

There are several bugs remained to be corrected in the future:

* Zombie nodes: The deregistration depends on the successful quit on slave. Therefore, if the slave node terminates somehow, the master node is unable to detect it forever. This will lead to a situation where other nodes fail to migrate the object to such node. And due to the design, the port which was previously used by accidently terminated process can never been reused. In order to mitigate this problem, a heart-beating query is required. The master can periodically send the heart-beating query to each slave node. If no response is received or exceptions are caught, master should update the status of each slave node.
* Class Not Found Exception: Since the hierarchy of all java files, when users launch a MigratableProcess, a full path from current directory to a specific class is required. More specifically, all the classes that implement MigratalbeProcess resides in the ./src/process, users should specify the full path, i.e. process.SomeMigratableProcess. This could be fixed by classloader later on.
* A safe way to quit for master node. A safe way to quit for master node hasn’t been implemented. At present, the master can only quit by using ctrl-c termination signal. This may lead memory leaks. Besides, due to the unsafe way to terminate master node, the slave node won’t be notified. And the migrate, query and quit cannot work properly.

1. **Build File**

The 15440Lab1.zip file is a Eclipse project folder. In order to build this file, you can import this folder into the Eclipse.

All the MigratableProcess java files resides in ./15440Lab1/src/process/. Additional MigratableProcess classes are required to put under this directory.

In order to export the .jar file, the main() function should be set to point to ProcessManager - 15440Lab1.

1. **Dependencies**

Operating System: Linux;

JRE: 1.7 or higher

File System: Distributed File System, e.g. AFS

1. **Examples**
   1. **Start Cluster**

* Start a master

1. Connect to an andrew machine. (e.g. $ssh gengf@linux.andrew.cmu.edu)
2. Copy the jar file to the directory.
3. Use $/sbin/ifconfig and write down its IP address.

(e.g. master-IP: 128.2.13.145)

1. Choose a port number(1024-65535). (e.g. master-port: 22024)
2. Starts master node: $ java -jar lab1.jar master 22024

* Start a slave

1. Connect to an andrew machine. (e.g. $ssh gengf@linux.andrew.cmu.edu)
2. Choose a port number(1024-65535). (e.g. local-port: 1234)
3. Starts master node: $ java -jar lab1.jar slave <master-IP > <master-port> <local-port> (e.g. $ java -jar lab1.jar slave 128.2.13.145 22024 1234)
4. If it launches successfully, the slave node name is shown which is used to globally identify each slave node.
5. Repeat step (1) – (2) to build more nodes if needed.
   1. **Commands supported**

At slave node, following commands are supported:

* quit: Slave node terminates.

Usage: >>quit

* ps: Poll all the IDs of MigratableProcess objects on this node and their statuses. It could be either running or finished.

Usage: >>ps

* query: Retrieve available nodes in the cluster.

Usage: >>query

* run: Launch a new thread to run MigratableProcess object.

Usage: >>run <Class Name> <args>

* migrate: Migrate a MigratableProcess object to another node.

Usage: >>migrate <PID> <Slave Node Name>

* 1. **Run FileCompress**

FileCompress object compress a source file to a .gzip type of file. In order to make enough time to migrate, a Sleep-Time (Non-negative)needs to be passed to indicate how long do you want the thread to sleep every 1KB data is compressed.

**Usage:**>>run process.FileCompress <Source-File> <Destination-File>

<Sleep-Time>

**Example:**>>run process.FileCompress compress.pdf compress.gzip 10

A test file named compres.pdf is provided. Now we can use command ps to see its PID and observe its status. Here is an example:

>>ps

PID Status

slave1-0 Running

Before migrate this process, check what are slave nodes that are available:

>>query

All slaves:

slave2

\*slave1

where \* stands for the slave node you are working on.

Suppose we want to migrate process with PID:slave1-0 to slave 2, we use:

>>migrate slave1-0 slave2

After Wait for several seconds during which the process is migrating, the prompt (>>)will show again.

On slave1, we use ps command:

>>ps

PID Status

where no process is found.

On slave2, we also use ps command:

>> ps

PID Status

migrate-slave2-0 Running

we can find that a migrated process is running on slave2.

We can verify our TransactionalFileIO functions correctly in two ways: On the one hand, the file size keeps increasing which means the file keeps writing at the end of the file. On the other hand, we can unzip the file and compare it with original file.

* 1. **Run WebCrawler**

WebCrawler writes down every URL it visits. WebCrawler starts a HTTP request to the URL we pass to it. It parses out all the HTTP link in the response file and save them for further requests. It traverses the links following Breadth-First-Search. The parameter of max line indicates the maximum lines it can write to the output file. It also acts as a condition decides whether a process finishes. The thread sleeps for 1s every time it request a page.

**Usage**:>>run process.FileCompress <Initial URL> <Output File> <Max Line>

(a http:// prefix is required for Initial URL)

**Example**:>>run process.WebCrawler http://www.cmu.edu output.txt 100

Following the previous commands, we can also migrate this process to other thread. As a verification of TransactionalFileIO works correctly, the file size keeps increasing and the total lines number matched to the Max Line parameter.