Project 1 Report

Architecture Design and Deployment Guide

Team: Menglong He (menglonh), Sidi Lin (sidil)

6/2/2014

Table of Contents

System Architecture Design and Pre-Requisition

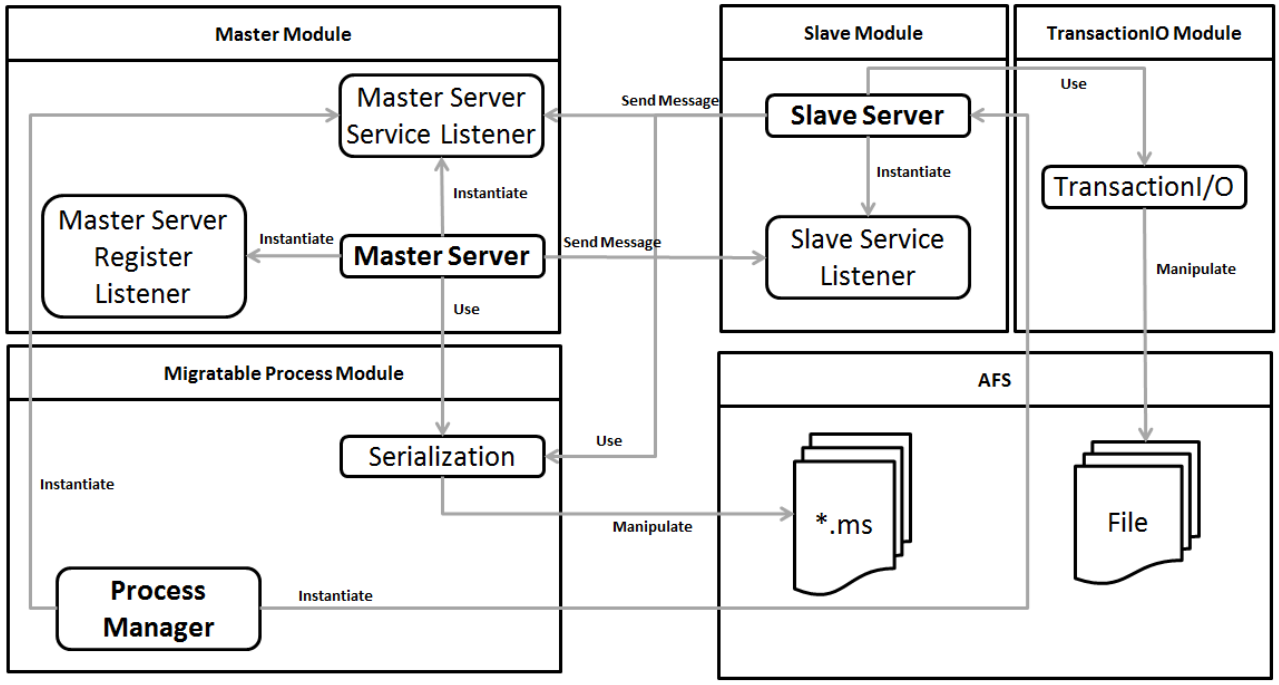
System Architecture Design

This system is developed by Menglong He and Sidi Lin as a course work for project 1 in course 15640/15440. The aim of this system is to emulate the migration of process between different nodes in specific distributed system which consists of two kinds of nodes: Master node and Slave node.

Master node is a command line console that runs on any one of the endpoint of the distributed system, which responsible for parsing the user's input, initializing/assigning and migrate processes to AFS or between Slave nodes, and monitoring the status of slave nodes and processes.

Slave node is a listening process that could be ran on any other system of the distributed system. It receives the command from Master node and process correspondence task per instructions. It does not have any user interaction function or flexibility. Basically slave node is the same as those in HDFS, which is mainly responsible for task processing and data storing. In this project, we are mainly focus on task processing.

The overall architecture of this system is as shown in following graph:



Process Manager

Process Manager is the entry point of this system. The whole system will be initiated by Process Manager – it instantiate Master Server and Slave server. Also it prints out message about starting and terminating these servers.

Master Server

Master Server in charge of all the process initializing, processing and migrating and it monitors status of all process and slave nodes. It will firstly setup the server socket and wait for any socket connection request and use one listener to register the slave nodes into the shared HashMap. Then it will provide user with a user-friendly command line environment to manage process and slave nodes. Each node (as well as process) will be assigned with a unique slave name (process id) automatically for monitoring case. All the tasks are initialized using reflection mechanism in the Master node. The initialized Migratable Process will be serialized into a file with suffix ".ms" and put the file into the common AFS which will be read by slave nodes for further operations. Whenever there is a change about process, master server will receive notification message from corresponding slave node so that all the registration information could be kept updated.

Master Server -- Listener

Service Listener is responsible for handling messages sent by slave nodes. Whenever a process is being processed, suspended, or finished, the slave node will notify the master node about the status of the process so that the master node could keep its information up to date. These update works will be finished by service listener. Hence, the master will setup a listener for each slave node when they build the socket connections.

Master Server -- Register Listener

Register Listener is responsible for managing the registration and connection information of slave nodes. Whenever a socket request is received, the listener will register it to node list and update the status of corresponding node. The master server will maintain a ConcurrentHashMap to record all the processes' updated info and slaves' info.

Slave Server

Slave Server is initialized by Process Manger and could be run on different endpoint than the master server. After started, it will try to connect to designate master node and then sent out its network configuration such as IP address and listening port number by setting up the socket connection. Then it will keep listening to that port for further instruction from the master node.

To start a process, slave server will need to read the process information which is sealed in a "*.ms" file on AFS using Serialization module. Also, the source and output files should also locates on AFS so that they could be shared among different slave nodes when the process is being migrated. Whenever there is a change of the process status, slave server will send a notification message to master server.

Slave Service Listener

Slave Service Listener is responsible for handing request sent by master node. These messages will include the operation to perform and corresponding process id. After receiving these messages it will inform slave server to perform designate operation.

Serialization

Serialization module is used to create/fetch the process files. Master/Slave nodes will use this module to manipulate the process file. Serialization module will ONLY create or delete "*.ms" file due to security need.

TransationalIO

TransationalIO module is used to read/write the content of source files and output files. It contains a RandomAccessFile object that could record the current point of file cursor, which enables the ability for process to resume from last processed point, and the path of file that is being accessed.

Configuration

Configuration contains two main parameters: Main Server Port and the File Directory. The main server port is used for all the slave nodes. They can use this port to setup the socket connection. The file directory is also a key parameter. It can provide the directory for the serialized file. Hence, you need to customize these two parameters before running the framework.

Pre-Requisition

There are some pre-requisition about this system needs to be specified:

1. Both master node and slave node are initialized with same class called ProcessManager but with different arguments.
2. There should be **ONLY** one master node in specific system.
3. The number of slave node is flexible. However, in order to process the initialized task, there should be at least **ONE** slave node.
4. In general circumstances, master node should be setup before any slave node. Even though slave node without a master node will keep trying to connect to master node, in this case, this system will not guarantee the succeed or the sequence of slave connection.
5. Master node's IP address and port number should be addressed when initializing slave node. This address is immutable and cannot be changed after setup.
6. Task will be fetch by Master node using reflection mechanism and initialize onto AFS. All tasks should extend base class MigratableProcess and implement correspondence methods.
7. All process will be stored onto AFS as a file with suffix ".ms".
8. All source files and target files of processes should be stored onto the common AFS.
9. There should be enough space of AFS to store the process files.
10. Only those tasks which are setup on AFS could be processed by slave node.
11. The "*.ms" file cannot be modify once created. So whenever the process is started and then suspended, slave server will delete the previous version of process file and created a new one.
12. Each process can only be processed by one slave node at the same time. Multiple processes can be run on one slave nodes at the same time.
13. Slave node will keep standing by even the task is finished. It will wait for new instruction from master node.
14. All the operations you do must follow the right instructions.
15. References: 1) https://github.com/ankitC/processManager; 2) https://github.com/yinxu-cmu/15640-p1

How to Run/Deploy

To deploy the system we need to deploy both the master server side and slave servers side. For deployment convenience, we merged both deployment packages into a single one so we do not have to identify packages when deploying to different nodes.

When you see the package of our uploading code, you will see the makefile document. You will do the following the serveral steps to make a jar.

1) Find the Configuration file in src/migratableProcess package and set the master node's port number (default: 15640) and the file directory.
2) Open your shell window and cd to the package that contains the makefile.
3) Type "**make new**" to create the directory of bin package that was put the class files.
4) Type "**make build**" to build all the codes according to the hierarchy of the classes.
5) Type "**make jar**" to build the "15640_DS_Lab1.jar" for the following operation.
6) If you want to rebuild the classes, you can type "**make rebuild**"; if you want to clean all the bin package you can type "**make clean**".
7) Although we provide the run function by the makefile, we do not suggest using this command. It can just run the server part.

When you get the jar package, you can deploy the jar into the Master node and Slave nodes.

Master Node Deployment

To deploy a master node, we need to first download the "15640_DS_Lab1.jar" file onto master node endpoint. After downloaded the project file, we can run the following command to setup master server:

java –jar 15640_DS_Lab1.jar

The above command will setup a master server on port 15640.

Slave Node Deployment

To deploy the slave node, we need to copy the "15640_DS_Lab1.jar" file onto the objective slave node endpoint. We can run the following command to setup slave server:

java –jar 15640_DS_Lab1.jar –s <Master Server's IP address>

Running Testing Tasks

Test Case 1: SUM

This task is used to plus hundreds of thousands of numbers and to print the all the intermediate numbers. In order to run this test case, you need to check the "numbers.txt" file has been put into the

right position to ensure it can be found. Before you run the test case, you have to ensure the master node and slave nodes are running well.

The operating steps as the following:

1) Start the Main Server and start 2 slave nodes. You can use the command "**ls slaves**" to check whether all the slaves' info has been registered into the master server;
2) Type "**task Sum numbers.txt output.csv**" to create the first task. After that you will see a new file has been created called "0.ms" which was stored the Process instance;
3) Type "**start 0 0**" to send a message to tell the slave 0 to run the process 0;
4) After running the process about 10 ~30 seconds, you can type "**migrate 0 0 1**" to tell the slave 0 to suspend the process 0 and tell the slave 1 to continue running the process 0;
5) You can also migrate the process 0 from slave 1 to 0 as you wish, but you must type all the commands in about 120 seconds since the whole process will finish about 2 minutes.
6) In the process, you can type the "**ls process**" to track the process's status and slave node.
7) If you encounter some unexpected exceptions you can type the "**help**" to check the command's specifications. Even you read the commands and still cannot get out of the exception, you can restart all the master node and slave nodes;
8) After the Process 0 finished, the process will be deleted. Next task will have the process Id 1.

Test Case 2: Find Hot Words

This task is used to list all the words that the frequency of the words in a given file has over the setting threshold. Of course we need to eliminate many stopwords. We use the Google newest stopwords file to digest some useful and popular stopwords. We put all these words into a HashSet in the StopWords java file. The threshold of the frequency you can set by yourself.

The operating steps as following:

1) Following the up test case, you need to type "**task FindHotWords books.txt output1.txt 50**" into the console;
2) Then you can type "**start 1 0**' to send the message to tell the slave 0 to run the process 1. Of course you can run many different process in one slave node. You can our two test cases and your own test cases to run the multiple processes in a slave node.
3) You can type "**migrate 1 0 1**" then the master server will send the message to the slave 0 to tell it suspend the process 1 and send the message to the slave 1 to run the process 1;
4) If the console tells you the process has executed successfully, you can terminate all the slave nodes and master node by typing "**terminated**".

Implemented/Unimplemented/Bugs

Implemented Functionalities

Process Manager

ProcessManager is the entry point of this system. It takes arguments to setup master server or slave servers on local machine.

Master Server

Master Server manages information of processes and slave nodes. It contains a command line GUI for user interaction and two threads that handle slave status and process messages. Master server also uses serialization module to create task on AFS and commanding slave nodes to process tasks as requested.

Slave Server

Slave Server receives command from master server and performs requested tasks. It receives commands from master server. Then it uses serialization module to fetch process object and transactional IO module to setup input/output file connection. Whenever there is a change of process status it will send out a slave message to master server.

MigratableProcess Interface

MigratableProcess is the base class of all processes. It requires subclasses to implement three methods: run(), suspend() and toString(). Also it stores process ID and has two status flags: isSuspended and isFinished, which is shared with slave server to monitor the status of the process.

Transactional I/O

Transactional IO contains a file pointer which demonstrates the visit point of last running of the process. Also it stores the path to target file. This module implements ObjectInput/OutputStream that can be used to read/write files in lines or bytes.

Serialization

Serialization module is used to create/delete process file on AFS. It will not edit existing process file but instead recreate it. It implements two methods Serialize() and Deserialize() to create/fetch process objects.

Testing Examples

We have implements two testing examples: Sum and FindHotWords

Sum

Sum is a task that takes each row as a number from source file and calculates the sum of all numbers so far. Then write it to the output file.

FindHotWords

FindHotWords is a task that summarizes all the word in a document and calculates occurrences of each word.

Unimplemented Functionalities/Bugs

Slave Nodes Workload Balancer

We could implement a workload balancer on master server to automatically migrate tasks assigned to slave nodes. This would help increasing the system performance.

Connection Status Manager

Slave server will keep trying to connect to master server after initialized. However, we did not implement a mechanism to monitor the status of connections. So if network condition turns worse, this system will not be able to reconnection to master server or recover from exceptions.

Communication Encryption

The communication between master server and slave servers are non-encrypted, which means the messages sent could be changed or intercepted by others and causing damage to the system.

Progress Monitoring

We could monitor the progress of tasks by setting up a private field for each process. So that we can know how much more time a process will need to finished and decide whether to migrate it or not.

System Robustness

We have handle most of the exceptions in this system. However, when there is a fatal or unexpected exception occurs, this system may not be able to recover from this exception thus the whole system will have to reboot.

System requirements, Dependencies and Software

System Requirements

Operating System:
Windows 7, MacOS or Linux

Runtime Environment:
Java 7 Update 51

Memory Size:
4GB or more. (Enough space for runtime data caching)

File System:
AFS or equivalent distributed file system.

Storage Size:
50 GB or more. (Enough space for process file and input/output data)