Wadoop Administrator Guide

Author: Guanjie Chen (guanjiec)
Wenhan Lu (wenhanl)

1. System Requirement

An cluster of all UNIX machines. (just like andrew machines)

2. Before You Start

First you should set the parameters in the src/config/Config.java. Configurable parameters shows as below. You need to recompile whole project after you change anything.

NUM_WHFS_REPLICA	Number of block replicas distributed accross machines
BLOCK_SIZE	Number of lines per block
NAMENODE_PORT	WHFS namenode port
DATA_PORT	The port used to transfer data
TASK_PORT	The port used to send task related message
HEARTBEAT_TIMEOUT	Heartbeat timeout in ms
DATANODE_FILE_PORT	DataNode file server port
LOCAL_BASE_PATH	Master node local file system base path
WHFS_BASE_PATH	WHFS base path in all nodes
MAP_RESULTS_FOLDER	The map and reduce result folder
MASTER_NODE	Hostname of master node
SLAVE_NODES	List of hostnames of slave nodes

You are free to use default parameters we provided in most of them, but please check out carefully the following points.

LOCAL_BASE_PATH: Make sure it is an existent path on the Master machine. All local file importing to WHFS is from this directory. Make sure you have **write permission** to this dir.

WHFS_BASE_PATH, MAP_RESULTS_FOLDER, WHFS_TEMP_PATH: These three path can be generated automatically. You don't need to create them manually, but you need to make sure they are in a directory that you have write permission, and they are not created by other users in AFS.
NUM_WHFS_REPLICA: Some number less than number of slaves will be reasonable, if not, our system will use number of slaves instead of user-specified parameter.
BLOCK_SIZE: Theoretically it can be set to any number. But our design has a limitation to set this parameter too small, which leads to too many splits. Our system takes use of split command in UNIX system, which has upper bound of 676 splits. So make sure you don't create more than this number of splits, or you will lose some data. Of course we can improve our system to support arbitrary BLOCK_SIZE when we have more time in future.
MASTER_NODE: Set to master hostname you are using. eg. unix1.andrew.cmu.edu
SLAVE_NODES: Set to slave hostnames you want to use. They are statically decided before your start your cluster.

Before start to run a mapreduce job, you should first put a local file in the **LOCAL_BASE_PATH** you set. Then you can import this file into whfs. For example, if you set the **LOCAL_BASE_PATH = "/tmp/mlocal",** you should make sure that the folder exists and the file you want to import to WHFS is in this folder.

For your convenience, we provide some test data in test/ directory. If you're using our default LOCAL_BASE_PATH. Run the following command to put test data to LOCAL_BASE_PATH.

```
cp test/* <path-to-your-master-local>  (cp test/* /tmp/mlocal )
```

Check in your LOCAL_BASE_PATH. If you can see **medium.txt, big.txt, dict.txt**, you're all set.

After config set properly, next thing to do is to compile source code to executables. Change directory to project base directory. (run "ls" to see src/, lib/ etc to verify, if there is no bin/, mkdir a new one in this name). Run make to **make** the project, after which you will see bin/ directory not empty. If you want to re-compile project. Run **make clear** and then **make**.

3. Start Cluster

Here is recommended configuration for our test.

```
public static final String MASTER_NODE = "unix1.andrew.cmu.edu";
public static final String[] SLAVE_NODES = {
        "unix2.andrew.cmu.edu",
        "unix3.andrew.cmu.edu",
        "unix4.andrew.cmu.edu"
};
// port for transferring data in mapreduce phase
public static final int DATA_PORT = 19782;

// port for sending task message
public static final int TASK_PORT = 18432;

// Number of replica per data block
public static final int NUM_WHFS_REPLICA = 2;

// Number of lines per block
public static final int BLOCK_SIZE = 100;

// WHFS namenode port
public static final int NAMENODE_PORT = 15826;

// Heartbeat timeout in ms
public static final int HEARTBEAT_TIMEOUT = 5000;

// DataNode file server port
public static final int DATANODE_FILE_PORT = 15827;

// Local base path
public static final String LOCAL_BASE_PATH = "/tmp/mlocal/";

// WHFS base path
public static final String WHFS_BASE_PATH = "/tmp/handinwhfs/";

// map and reduce result file folder
public static final String MAP_RESULTS_FOLDER = "/tmp/handinfinal/";

// WHFS temp path
public static final String WHFS_TEMP_PATH = "/tmp/handinwhfstemp/";
```

In this configuration, we started a one master three slaves architecture. NUM_WHFS_REPLICA is set to 2 which is one smaller than cluster size. To run this test, we strongly recommend to open **five** consoles in your computer, **two** for master node and one for each slave node.

The reason we need **two** consoles for master node is for monitoring convenience. One will be used as monitor console. We will be writing logs when we run jobs. You can use “tail -f log” to monitor job status on second console. And after job is finish, you can also check output file in second console without stop master process in the first console.

Now let's start machines. First you **must** ssh to machines specified in config file, in this case is unix1.andrew.cmu.edu for master, unix(2,3,4).andrew.cmu.edu for slaves. Change directory to <project-path>/bin/ on all consoles except monitoring one.

Start master on one master console (Don't run it on both master console) by

java Master

Start slaves on slave machines by:

java Slave <master_hostname> (unix1.andrew.cmu.edu in this case)

Note: Now there is a limitation on our static implementation. You **must** run above command in slave machine in the same order you defined in config file. (In this case order is unix2, unix3, unix4). Otherwise common functions of our system works well, but it will not function normally when you test failure recovery.

Each time a slave node is successfully connected to the master node, you will see a message in the master node indicates that.

Here is command reference for master console.
For WHFS:

listfile	List all files under WHFS
listnode	List all nodes connected with master
replica	Print replica information (data block => node map)
nodes	Print what data blocks every block have (node => data blocks map)
import <local_file_path> <whfs_file_path>	Import a file from local file system to WHFS

For MapReduce

mprun <MR class name><whfs_file_path>	Run a mapreduce job
concurrent <MR className 1> <MR className 2> <Job1 input whfs filename> <Job2 input whfs filename>	Run two MapReduce jobs concurrently for your testing convenience.

4. Import File from Local to WHFS

After all the slaves have successfully connected to the master node, the next step is to import file from local file system to WHFS

To do so, type on master console:

```
import <Local_path> <whfs_path>
```

In our example, it can be:

```
import medium.txt medium
```

to import local file testFile.txt to the whfs and the name of the file in whfs is “test” (Please make sure that the name of file in whfs is unique!)

You can see a message “import finished” when you import file successfully.

5. Run Basic MapReduce Example

Before start running MR job, we open log file in order to monitor its status in monitor console. Change directory to **<project_base>/Log** and then run:

```
tail -f log
```

Then we go back to master console, let’s make sure running a basic WordCount works well.

```
mprun WordCount <whfs_filename>
```

In our case:

```
mprun WordCount medium
```

After you launch a job, you will see the process messages in each node. You can see message update in monitor console about status of all tasks. When all the tasks complete, the master node will show that all the result is written in the output file. And you can check the output file in the **MAP_RESULTS_FOLDER** you set. We recommend to do this in monitor console instead of stopping running master process in master console. Use ctrl + c to exit tail monitor and take a look in **MAP_RESULTS_FOLDER.** Output file name is identified by input file name. In our case, it should be MP_Result_small.

Don’t be surprised when you see result is not completely sorted like Hadoop. Each reducer returns a sorted result and we don’t sort them after merge because we don’t think it is necessary.

Note: Because output is identified by input file name, you cannot run different using same input file in WHFS, otherwise new output will overwrite previous one.

You can also test another example “anagram” use the same way. This example output all anagrams in a dictionary. Note that input file of anagram job must follow exactly one word per line (dict.txt as we give).

On monitor node:

```
tail -f <project_base>/Log/log
```

On master main:

```
import dict.txt anagram
mprun Anagram anagram
```

6. Run Concurrent MapReduce

Our mapreduce facility can support run two jobs concurrently and we provide you a command **concurrent** to test this.

```
concurrent <MR className 1> <MR className 2> <Job1 input whfs filename>
<Job2 input whfs filename>
```

Just like the above you should make sure the all the files you want to run are already in whfs. Here is the whole thing.

```
import medium.txt test1
import dict.txt test2
concurrent WordCount Anagram test1 test2
```

The first keyword concurrent tells the system you want to run two jobs. The two jobs will run concurrently. After job finished, check **MAP_RESULTS_FOLDER/**MP_Result_test1 and MP_Result_test2 on monitor console.

7. Testify Failure and Recovery during MapReduce job

Our mapreduce facility can also support recovery from failure, which is when one datanode to disconnected from the namenode, the namenode will find this failure and rerun the mapreduce job.

We use heartbeat to achieve this. When the datanode does not respond in some time (default 5 seconds in config), we will treat it as disconnected node. And then we will pick the job from job list and resubmit the job to enable it to run another time.

Because we need to run a job which last a long time to test it, we need to change configuration a little bit.

- Use ctrl+c to stop all processes. Ignore any broken pipe exception in this process.
- In main master node, change one parameter in <project_base>/src/config/Config.java BLOCK_SIZE to 10000
- Clean previous compilation and compile again. (make clean, then make under <project_base>)
- cd bin
- Restart master (java Master)
- Restart slaves in order of configured order. (unix2, unix3, unix4 in our default config)

Note again slave starting order is important in our implementation.

Then type following command in master console:

```
import big.txt big
mprun WordCount big
```

When running this large job which may takes several seconds (still short), you can manually disconnect one datanode by ctrl + c (quickly). And the rerunning process can be seen from the master node.

When you see something like:

```
Some slave disconnected
DataNode 128.2.13.134 timeout
```

It means one datanode is disconnected from master node.
And when you see:

```
----------Start rerunning!----------
```

It means the job is rerunning.

Check in monitor console under **MAP_RESULTS_FOLDER/**MP_Result_big after rerunning finished.

8. Write in the last

Thanks for reading this long report. We tried our best to make this document as clear as possible. We think we designed a thorough test for you. And all cases are passed on our end in andrew machines. However this document may have typo or flaws, if you see something not working, please contact us with wenhanl@andrew.cmu.edu if convenient.