CS553 Programming Assignment #2

WordCount on Hadoop/Swift

Instructions:

- ***Due date: 11:59PM on Thursday, 10/23/14***
- ***Maximum Points including extra credit: 160%***
- *You should work in teams of 3 for this assignment.*
- *Please post your questions to the Piazza forum.*
- *Only a softcopy submission is required; it must be submitted to "Digital Drop Box" on Blackboard.*
- *For all programming assignments, please submit just the softcopy; please zip all files (report, source code, compilation scripts, and documentation) and submit it to BB.*
- *Name your file as this rule: "PROG#_LASTNAME1_LASTNAME2_LASTNAME3.{zip|tar|pdf}". E.g. "Prog1_Raicu_Li_Wang.tar".*
- *Late submission will be penalized at 10% per day (beyond the 7-day late pass).*

1. Introduction

The goal of this programming assignment is to enable you to gain experience programming with:

- Amazon Web Services, specifically the EC2 cloud (http://aws.amazon.com/ec2/)
- The Hadoop framework (http://hadoop.apache.org/)
- The Swift parallel programming system (http://swift-lang.org/main/)

Sign up for an account on Amazon Web Services. We have secured sufficient funds for each student to use $100; instructions on how to use this credit has already been distributed by the TAs through email. If you have not received the $100 credit code, please send email to the TAs at cs553-f14@datasys.cs.iit.edu.

2. Your Assignment

This programming assignment covers the WordCount application implemented in 3 different ways: Java, Hadoop, and Swift. This assignment will be broken down into eight parts:

1) **Virtual Cluster (1-node) [5 points]:** Setup virtual cluster of 1 node on Amazon EC2
2) **Shared-Memory WordCount [10 points]:** Implement the Shared-Memory WordCount application in your favorite language (without using Hadoop or Swift) and measure its performance on 1 node on a c3.large instance (from here on, this will be called Share-Memory WordCount); you should make your Shared-Memory WordCount multi-threaded to take advantage of multiple cores
3) **Virtual Cluster (16-nodes) [10 points]:** Setup virtual cluster of 16 node on Amazon EC2 (using the same AMI from step #1)
4) **Hadoop [20 points]:** Install Hadoop on 16 nodes (including the HDFS distributed file system)
5) **Hadoop WordCount [20 points]:** Implement the Hadoop WordCount application, and evaluate its performance on 1 node and 16 nodes. You should be doing strong scaling experiments as you scale up from 1 node to 16 nodes.
6) **Swift [20 points]:** Install Swift on 16 nodes

7) **Swift WordCount [20 points]:** Implement the Swift WordCount application, and evaluate its performance on 1 node and 16 nodes. You should be doing strong scaling experiments as you scale up from 1 node to 16 nodes.
8) **Performance [20 points]:** Compare the performance of the three versions of WordCount (Shared-Memory, Hadoop, and Swift)) on 1 node scale and explain your observations; compare the Shared-Memory performance of 1 node to Hadoop and Swift WordCount at 16 node scales and explain your observations.
9) **Sort on Shared-Memory, Hadoop, Swift, and MPI [35 points]**: Implement and evaluate sorting across four approaches, shared memory in your favorite language (5 points), Hadoop (5 points), Swift (5 points), and MPI (20 points).

2.1. Virtual Cluster (1-node) [5 points]

Install your favorite Linux distribution in a virtual machine on Amazon EC2. If you want to use a pre-created image from Amazon, that is fine as long as you specify what image you used. Make sure ssh is enabled on your Linux install. You will need to setup several applications, such as Java [**3**], ANT [2] (works great as a Makefile for Java programs), Hadoop [4, 5, 8], and Swift [cite]. You should use "spot instances" as they are less expensive, as well as "c3.large" instance types (see http://aws.amazon.com/ec2/instance-types/ for more details; also http://aws.amazon.com/ec2/pricing/ for pricing).

2.2. Shared-Memory WordCount [10 points]

WordCount [1] is an application which counts the occurrences of each word in a given set of text files. You are to implement the WordCount application in your favorite language, without MapReduce/Hadoop or Swift. WordCount should be multi-threaded (to process large files concurrently), and should only run on 1 virtual node. You can develop your code on any machine, but your performance evaluation must be done on Amazon EC2 on a "c3.large" instance. Measure the time to execute the WordCount application on the input data [6, 7] on 1 node. Vary the number of threads from 1 to 8, to find the best performance. Save the Java WordCount application output when processing the 10GB dataset to wordcount-java.txt.

2.3. Virtual Cluster (17-nodes) [10 points]

Setup virtual cluster of 17 nodes (you may find it useful to have 1 node for control, and 16 nodes for compute/storage) on Amazon EC2 (using the same AMI from step #1). You may find useful (as you scale up to multiple VMs) to install a shared file system (NFS) between the virtual machines, but it is not required. Alternatively, you may find useful to install S3FS to convert S3 into a POSIX-compatible shared filesystem.

2.4. Hadoop [20 points]

Install Hadoop including the HDFS distributed file system. You may want to configure Hadoop to run the job tracker and filesystem metadata service on separate nodes, leaving 16 nodes available to run workers for map and reduce tasks. Please follow instructions at [5, 8] on how to install Hadoop.

2.5. Hadoop WordCount [20 points]

WordCount [1] fits very well with the MapReduce programming model making it a great example to understand the Hadoop Map/Reduce programming style. It consists of three main parts:

- Mapper
- Reducer
- Main driver program

Mapper

The input text files will be split by the Hadoop framework as pairs <line_number, line_of_text>, and these pairs are passed to each map task. It then splits the "line_of_text" into tokens separated by whitespaces, and emits an intermediate key-value pair of <word, 1>.

```
void Map (key, value)                    {
    for each word x in value:
    output.collect(x, 1);
}
```

Reducer

A Reducer collects the intermediate key-value pairs from multiple map tasks and assembles a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

```
void Reduce (keyword, <list of value>)                                                    {
        for each x in <list of value>:
            sum+=x;
        final_output.collect(keyword, sum);
}
```

Main driver program

The Main program configures and run the MapReduce job. We use the main program to perform basic configurations such as:

- **Job Name** : name of this Job
- **Executable (Jar) Class**: the main executable class. For here, WordCount.
- **Mapper Class**: class which overrides the "map" function. For here, Map.
- **Reducer**: class which override the "reduce" function. For here , Reduce.
- **Output Key**: type of output key. For here, Text.
- **Output Value**: type of output value. For here, IntWritable.
- HDFS File Input Path
- HDFS File Output Path

In this project, you will need to implement the mapper, reducer and main driver program shown above. Please download your input text file [6, 7] to test your WordCount application. Make sure that HDFS is configured to use a local file system and not a shared file system (e.g. NFS, /home/userid/). On "c3.large" instances you will have 32GB of local disk per instance, please use this disk for HDFS.

In case you ran your Hadoop WordCount on a single Hadoop node first for sanity check, you will likely have to modify these configuration files from the 1 virtual node case to the multi-node cases:

1) conf/master
2) conf/slaves
3) conf/core-site.xml
4) conf/hdfs-site.xml
5) conf/mapred-site.xml

You are to write a paragraph outlining the function of each file, and what modifications you had to make to go from 1 node to multiple nodes.

Hints:

1) What is a Master node? What is a Slaves node?
2) Why do we need to set unique available ports to those configuration files on a shared environment? What errors or side-effects will show if we use same port number for each user?
3) How can we change the number of mappers and reducers from the configuration file?

Save the Hadoop WordCount application output when processing the 10GB dataset to wordcount-hadoop.txt.

2.6. Swift [20 points]

Install Swift on 17 nodes (including the POSIX layer over S3 via FUSE, as Swift expects a shared POSIX filesystem between all the compute nodes).

The repo where you will find Swift and relevant materials are can be found at: https://github.com/yadudoc/cloud-tutorials.git. There is some documentation on the github page itself.

There two ways to go after you get the Amazon access key and access key id from AWS console, and depending on what your source OS is. We recommend using popular Linux distributions such as Ubuntu and Fedora though.

1. You have a Linux / Mac system
   - Install python-pip, apache-libcloud
   - git clone cloud-tutorials
   - Configure and start the cluster
2. You have a Windows machine
   - Go to AWS console, and setup a keypair
   - Launch an instance named launchpad, with the keypair
   - scp the credentials.csv and keypair to launchpad
   - Login to launchpad
   - install python-pip, apache-libcloud, git
   - git clone cloud-tutorials
   - Configure and start the cluster
3. Connect to headnode
4. Run tutorials
   a. The simplest example is in the “swift-cloud-tutorial/cloud-cat” as a distributed “cat”
5. Run wordcount (after implementing it, see section 2.7)
6. Stop resources

2.7. Swift WordCount [20 points]

Implement the Swift WordCount application. Save the Swift WordCount application output when processing the 10GB dataset to wordcount-swift.txt.

2.8. Performance [20 points]

Compare the performance of the three versions of WordCount (Shared-Memory, Hadoop, and Swift)) on 1 node scale and explain your observations; compare the Shared-Memory performance of 1 node to Hadoop and Swift WordCount at 16 node scales and explain your observations. You should be doing strong scaling experiments as you scale up from 1 node to 16 nodes, using the provided input data sets [6, 7].

Hint: Since your c3.large instance will have 2 virtual cores, you will configure Swift and Hadoop to run 2 workers (for Swift) and 2 mappers/reducers (for Hadoop). You can change the total amount of mapper and reducer by editing the configuration files "conf/mapred-site_template.xml"and "conf/mapred-site.xml".

Draw an execution line chart and a speed up line chart for 1 node and 16 node cases, for Java, Hadoop, and Swift WordCount. For Swift and Hadoop, compute two different speedups, using different base cases; one speedup (speedup-java) should be relative to the Java WordCount performance (note that you might get a speedup less than 1); the second speedup (speedup-swift & speedup-hadoop) should be relative to the Swift or Hadoop performance at 1 node scale respectively (this should be a number greater than 1).

What conclusions can you draw? Which seems to be best at 1 node scale? How about 16 nodes? Can you predict which would be best at 100 node scale? How about 1000 node scales?

2.9. Sort on Shared-Memory, Hadoop, Swift, and MPI [35 points]

Implement and evaluate sorting across four approaches, shared memory in your favorite language (SharedMemorySort - 5 points), Hadoop (HadoopSort - 5 points), Swift (SwiftSort - 5 points), and MPI (MPISort - 20 points). Your sorting application could read a large file and sort it in place (your program should be able to sort larger than memory files). Use the file generator at [9] to generate a 10GB dataset. For MPISort, you will have to setup MPI on your virtual cluster of 16 nodes, and implement MPISort. Measure the performance of the Shared-Memory sorting (including reading data from disk, sorting, and writing data to disk), Hadoop Sort at 1 node and 16 node scales, the Swift Sort at 1 node and 16 node scales, and the MPISort at 1 node and 16 node scales. Please save the first 1MB of output data to include in your final submission, such as "sort1MB-sharedmemory.txt", "sort1MB -hadoop.txt", "sort1MB -swift.txt", and "sort1MB-mpi.txt".

3. Deliverables

You are to write a report (prog2_report.pdf). Your report should include the authors' information, and a brief overview of each team member's contributions.

Add a brief description of the problem, methodology, and runtime environment settings. Discuss the installation steps you took to setup your virtual cluster. Please include any difficulties you might have incurred in your setup of the virtual cluster. Also, include a detailed description of what OS you used (Linux distribution, kernel), what ANT version, what Java version, what Hadoop version, what Swift version, and what MPI version.

You are to draw graphs showing execution time data and speedup data. Please explain your results. Please include a comparison between the SharedMemory WordCount compared to the Hadoop WordCount and Swift WordCount for 1 node and Hadoop and Swift at 16 nodes. What about Sorting across the 4 different approaches? Can you explain the difference in performance?

You are required to turn in a zipped or tarred package named as "prog2_lastname1_firstname1_lastname2_firstname2.{zip|tar}" on BlackBoard.

1) Source code and configuration files for both the single node and multiple node cases; do not include the dataset files
2) A compiled jar executable
3) A readme.txt file with explanations of code, which file contains what, and the commands needed to compile and execute the different scenarios
4) Output files "wordcount-java.txt", "wordcount-hadoop.txt", "wordcount-swift.txt" & "sort1MB-sharedmemory.txt", "sort1MB -hadoop.txt", "sort1MB -swift.txt", and "sort1MB-mpi.txt"
5) Report file "prog2_report.pdf"

4. References

[1] http://hadoop.apache.org/docs/r0.20.2/mapred_tutorial.html

[2] http://mirrors.axint.net/apache/ant/binaries/apache-ant-1.8.2-bin.tar.gz

[3] http://www.oracle.com/technetwork/java/javase/downloads/index.html

[4] http://hadoop.apache.org/common/docs/r0.20.2/quickstart.html

[5] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/ClusterSetup.html

[6] 10GB dataset: https://s3.amazonaws.com/cs-553/wiki10gb.xz or https://www.dropbox.com/s/5s68zj44xhijvmp/wiki10gb.xz?dl=0

[7] Small dataset: https://s3.amazonaws.com/cs-553/small-dataset or https://www.dropbox.com/s/kq85q0ljcovp57c/small-dataset?dl=0

[8] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/SingleCluster.html

[9] http://www.ordinal.com/gensort.html

[10] Swift; http://swift-lang.org/docs/index.php

[11] Swift Tutorial; https://github.com/yadudoc/cloud-tutorials.git