Group Name: A.F.K.
Participants: Tuan **A**nh Hoang Vu (hvtuananh@gmail.com)
Fernando Seabra Chirigati (fernando.chirigati@gmail.com)
Kien T. Pham (kienpt.vie@gmail.com)
Task: 4 - Extracting metadata from Wikipedia pages

Responsibilities from each participant

- Tuan Anh: Writing scripts to uncompress the input files, to setup Amazon EC2 and Hadoop, and to run the map-reduce code.
- Fernando: Writing a FileInputFormat class to correctly split the Wikipedia input files into chunks of pages.
- Kien: Writing the mapper for the task.

GitHub repository

https://github.com/kienpt/wikipedia_statistic

Metadata Output

We have two different sets of output files, which only differ in the number of output files generated.

1. s3n://cs9223-results/2012-11-21-074821
   https://s3.amazonaws.com/cs9223-results/2012-11-21-074821/part-0xxxx
   with xxxx from 0000 to 1432 (1433 output files)
2. s3n://wikimetadata/2012-11-25-222927
   https://s3.amazonaws.com/wikimetadata/2012-11-25-222927/part-00xxx
   with xxx from 000 to 295 (296 output files)

(Note: the size of each output file in the second experiment is around 55 MB, which is closest to the HDFS block size -- 65 MB, so this is the most appropriate result for using in phase 2).

PS: We did not use a reducer, as we can explore the already splitted output files for the next phase of the project. Also, using only one reducer to merge the files would lead to a bottleneck problem. Note that the reducer would be very simple, only to merge the files.

Instructions to set up the environment and to configure AWS

- Install ec2-api-tools package
  - Ubuntu
    - sudo apt-get install ec2-api-tools
  - Mac
    - Download the package from this website

        - Put ec2-api-tools/bin in the PATH variable
            - export PATH=$PATH:/path/to/ec2-api-tools/bin/
        - Set up EC2_HOME variable
            - export EC2_HOME=/path/to/ec2-api-tools/
- Create an EC2 key pairs with name '**cs9223**' (the name is important!), chmod to 400 (this is also important!), download private key and place it under wikipedia_statistic/ec2/bin/ directory.
- Generate access key and secret key and put them at the end of ~/.profile (Ubuntu) or ~/.bash_profile (Mac) -- other distributions might have different configurations

```
# set aws variables
export AWS_ACCESS_KEY="access_key"
export AWS_SECRET_KEY="secret_key"
```

- Change amazon account number, access key and secret key (generated above) in wikipedia_statistic/ec2/bin/hadoop-ec2-env.sh.sample, and rename it to hadoop-ec2-env.sh
- Change access key and secret key in wikipedia_statistic/ec2/bin/hadoop-ec2-init-remote.sh.sample, and rename it to hadoop-ec2-init-remote.sh

Instructions to run our code

- To setup cluster:
    - `cd wikipedia_statistic/ec2`
    - `bin/hadoop-ec2 launch-cluster <cluster-name> <number-of-slaves>`
      (master and slaves will be launched as *small* instances -- change type of instance in wikipedia_statistic/ec2/bin/hadoop-ec2-env.sh if necessary)
- To run phase 1
    - `bin/hadoop-ec2 exec <cluster-name> ec2/bin/cs9223-phase1.sh`

Configuration in EC2

Our idea was to use 16 slave nodes + 1 master, which is almost the maximum number of instances that can run simultaneously on EC2 (20 instances). Initially, we used the first generation small instance for all the nodes, because we thought the cost-benefit was good. However, it was taking a long time to initialize and set up Hadoop, and the master was always going down. We decided to keep using the small instances for the slaves, but getting a large instance for the master -- then, it worked. The maximum number of map tasks per node was configured to 8 (note that this is just an estimative to Hadoop).

We chose three different configurations with respect to the number of Wikipedia pages per split (more on this later):

(a) 20,000 pages per split, which created 1433 map tasks;

(b) 200,000 pages per split, which created 143 map tasks;

(c) 100,000 pages per split, which created 296 map tasks.

The Hadoop setup phase (creating stream, reading all input to calculate the total number of map tasks and splitting the input files) took 1 hour and 7 minutes to complete in (a), and around 1 hour and 1 minute in (b) and (c), and the running time was of 1 hour and 12 minutes in (a), 55 minutes in (b) and 1 hour and 5 minutes in (c). It is interesting to notice the overhead in setting up Hadoop -- the setup time was almost the same as the running time in both configurations. We believe that the main reason for this long setup is the overhead in splitting all the input data to distribute among the map tasks.

Discussion

Since Python is a nice script language, and we find it easier to use for the task of parsing files, we decided to use Hadoop Streaming. So, in our case, the map code was implemented in Python.

One important consideration with respect to any Wikipedia-related task is that mappers need to receive whole Wikipedia pages. So, we had to make sure that, when splitting the input files, the boundaries of the Wikipedia pages were being respected -- a page cannot be splitted between two map tasks. Our initial idea was to use WikiHadoop, which has a special FileInputFormat class to process the compressed XML dumps of Wikipedia. However, WikiHadoop works for Hadoop versions 0.21, 0.22 and 0.23, and we were using version 1.0.3, which is more recent and stable. Also, even with these older versions, we could not make it work properly: the FileInputFormat class was not able to uncompress the dumps. Besides WikiHadoop, we also tried StreamXMLRecord, a record reader used to process XML document, but it was generating duplicate records in the output files.

Given these issues, we decided to implement our own FileInputFormat, named *WikipediaInputFormat*. This class automatically splits the input files into chunks of Wikipedia pages, and it also has a configuration parameter to set up the number of pages per split (*mapred.page.input.format.pagespermap*). This class, however, is not able to uncompress the dumps; we uncompressed the input files in EC2 before running the map-reduce code -- the uncompressed data was placed in S3 and read directly from S3.

Although *WikipediaInputFormat* worked in Hadoop 1.0.3, it did not work very well in Hadoop 1.1.0, which was the version we initially had in the Amazon instances. We could not find the reason for this different behavior, but it seems that there is a difference in the implementation of the class FileSplit, which creates the splits in the Hadoop framework, between the mentioned versions. As we did not find the solution, we decided to create the Amazon instances with Hadoop 1.0.3, instead of using the version 1.1.0. It is important to note here that, for our task, using either version is fine.

Furthermore, we also found out a difference between these versions on the way Hadoop works with the s3n URI scheme. Hadoop 1.1.0 handles any s3n URI with no problems. For example, executing the command “bin/hadoop fs -ls” on *s3n://cs9223/enwiki-20121001/* returns exactly 27 input files. However, there is a small difference in Hadoop 1.0.3 -- the command above returns 28 items instead, and the first item is *s3n://cs9223/enwiki-20121001/enwiki-20121001*, that actually links to itself. As a result,

executing a “-get” command on *s3n://cs9223/enwiki-20121001/* will run forever. To fix this, we had to use a file filter in order to filter out the bogus link.