Big Data 4

1st Programming Assignment

Introduction

The goal of this assignment is to familiarize the students with the design, implementation and performance testing of Big Data crunching tasks using Hadoop/MapReduce. You will be required to design and implement algorithms for parsing, filtering, projecting, and transforming data, so as to process a set of user-supplied queries, over a relatively large dataset, executing your code on a shared Hadoop cluster.

Dataset and Infrastructure

You will be working on a parsed version of the complete Wikipedia edit history up to January 2008[1]. This is a single large text file (around the 300GB mark), in a tagged multi-line format. Each revision history record consists of 14 lines, each starting with a tag and containing a space-delimited series of entries. More specifically, each record contains the following data/tags, one tag per line:

- `REVISION`: revision metadata, consisting of:
  - `article_id`: a large integer, uniquely identifying each page.
  - `rev_id`: a large number uniquely identifying each revision.
  - `article_title`: a string denoting the page's title (and the last part of the URL of the page).
  - `timestamp`: the exact date and time of the revision, in ISO 8601 format; e.g., 13:45:00 UTC 30 September 2013 becomes `2013-09-12T13:45:00Z`, where `T` separates the date from the time part and `Z` denotes the time is in UTC.
  - `[ip:]username`: the name of the user who performed the revision, or her DNS-resolved IP address (e.g., `ip:office.dcs.gla.ac.uk`) if anonymous.
  - `user_id`: a large number uniquely identifying the user who performed the revision, or her IP address as above if anonymous.
- `CATEGORY`: list of categories this page is assigned to.
- `IMAGE`: list of images in the page, each listed as many times as it occurs.
- `MAIN, TALK, USER, USER_TALK, OTHER`: cross-references to pages in other namespaces.
- `EXTERNAL`: list of hyperlinks to pages outside Wikipedia.
- `TEMPLATE`: list of all templates used by the page, each listed as many times as it occurs.
- `COMMENT`: revision comments as entered by the revision author.
- `MINOR`: a Boolean flag (0|1) denoting whether the edit was marked as minor by the author.
- `TEXTDATA`: word count of revision's plain text.
- An empty line, denoting the end of the current record.

In order to execute and test your implementations, you will be given access to a cluster of computers, consisting of a number of nodes. The dataset will be provided to you on the Hadoop cluster, where your assignments will also be tested and graded. The exact path, security credentials,

[1] Source: *http://snap.stanford.edu/data/wiki-meta.html*

as well as detailed instructions on how to access the cluster and execute your code, will be handed out in the course tutorials.

Assignment details

Part A:

You will have to design, implement and test a set of programs dealing with the data outlined above. More specifically, you must implement MapReduce programs able to process the following types of queries over the dataset:

1. Given a time interval (in the form of two ISO 8601 formatted timestamps, earlier one first), print all `article_id`'s for the pages modified in this interval, along with the number of modifications performed on each of them and a space-separated list of the actual `rev_id`'s; i.e.,
   ```
   article_id_i 2 rev_id_a rev_id_b
   article_id_j 1 rev_id_c
   article_id_k 3 rev_id_d rev_id_e rev_id_f
   …
   ```
   The output should be sorted on `article_id`, with each list of `rev_id`'s also being sorted.
2. Given a time interval (in the form of two timestamps, earlier one first) and a number k, print all `article_id`'s for the k pages with the highest number of modifications in the given time interval, plus the actual number of modifications for each page. The output should be sorted first by the number of modifications and then by `article_id`; i.e.,
   ```
   article_id_l 21
   article_id_n 19
   article_id_m 13
   …
   ```
3. Given a timestamp (same format as the timestamps in the dataset), print all `article_id`/`rev_id` pairs for the revisions that were "current" at that point in time, plus the timestamp of the corresponding modification, all in a space-separated format. That is, for each article, you must find the single revision whose timestamp is closest but not past the timestamp supplied by the user. The output should be sorted by `article_id`; i.e.,
   ```
   article_id_i rev_id_a 2012-01-01T11:22:33Z
   article_id_k rev_id_e 2011-11-22T00:11:22Z
   article_id_m rev_id_h 2011-09-31T22:33:44Z
   …
   ```

In all cases, input must be given to your programs from the command line; for example, for query 1 the invocation will be along the lines of:

```
java … your.package.Main 2013-08-01T13:45:23Z 2013-09-13T09:18:52Z
```

In all cases where sorting is required, the output should be in ascending numeric order.

Part B:

For each of the queries outlined above, you must perform an experimental evaluation of the performance of your implementations. Decide on a set of intervals/time points and execute your code multiple times for each, measuring the mean value and standard deviation of (a) the query

processing time, (b) the number of bytes read from HDFS, and (c) the number of bytes transferred over the network.

Returns

You are urged to work in 2-person teams. The deadline for assignment turn-in is **Friday, 13 February 2015.** You must hand in a single zip/tar file, including:

- The source code for the implementation of the query processing algorithms.
- A short report in PDF format, outlining your design decisions in building the above, any hardships you faced during the implementation and execution of your code, lessons learned, as well as the results of your performance tests and findings for Part C.

Submissions should be done via Moodle.