INF344, Télécom ParisTech

HBase

Pierre Senellart (pierre.senellart@telecom-paristech.fr)

13 June 2016

The purpose of this lab session is to get familiar with the use of the HBase distributed storage system, and with the use of MapReduce jobs to extract data from HBase, and to produce data to load into HBase in bulk. We reuse the inverted index application, but this time rely on an HBase-located corpus.

1 HBase installation and Hadoop cluster

The corpus we use is in the `simplewiki` table of HBase, and you have read access to this table. You have read and write access to one table within HBase, whose name is `inf344_`*`login`* where *`login`* is your Unix login. The only family defined for this table is `index`. You also have read-and-write access to one HDFS directory, `/user/inf344/`*`login`* where *`login`* is your Unix login.

You must make sure to access HDFS or HBase from an account that has for name your Unix login so as not to get permission errors; it is not an issue if you launch commands from one of the computing servers, or if you run your Java program from one of the lab computers. But if you want to execute code from your own machine, in addition to installing all prerequisite software, you will also need to make sure your account name matches your Unix login.

You can have access to Télécom ParisTech's Hadoop cluster from the computing servers `lame11`, `lame12`, `lame13`, `lame14`, `lame15`, or `lame16`. Just choose randomly one of these machines, and log on to it using ssh within a terminal. Your files remain accessible whatever machine you choose.

Once logged in on a terminal on one of the computing servers, run the following two commands to set up the Hadoop environment:

```
. /home/local/bin/hadoop_env
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$(hbase mapredcp)
```

(do not forget the leading period in the first command).

You will then be able to use the following commands:

hadoop fs -ls /user/inf344/*login* lists the file in your HDFS directory; you can replace "ls" with other standard Unix commands, such as "rm", "mkdir", "cat", etc.

hadoop jar myjar.jar MyClass to run a MapReduce job packaged within a JAR file (see further how to obtain this JAR file) and execute the MyClass file within

hadoop job -list to see all active MapReduce jobs

hadoop job -kill job to kill an active MapReduce job

hbase shell to enter an interactive HBase shell; type "help" to get basic information and "list" to see accessible tables. The most useful command within this shell is "get", to get a cell by row, family, qualifier in an HBase table; you can also use "count" to count the numbers of row in an HBase table.

You have access to an interface to see current and recent MapReduce jobs at `http://lame11.enst.fr:19888/jobhistory`.

2 Baseline

A skeleton Eclipse project is available for download from the course Web site. You are free to use another Java development environment; if you do not use Eclipse, you will need to make sure both the `config/` directory and all JARs in the `lib/` directory are in the class path.

The skeleton project contains the following JAVA source files:

InvertedIndex.java This is the main part of the program to implement and the only file you should modify. When submitting your solution to the submission server, you only need to submit this file. *TODO* annotations indicate which parts of the file need to be filled in.

MyConfigurationFactory.java A utility class that generates an appropriate Hadoop/HBase configuration.

DeleteAll.java A utility class that you can use to delete all rows from your HBase table. **If you use to empty your HBase table, use this class; do not under any circumstances use the HBase shell "drop" command, you would not be able to recreate a new table.**

PairTextDoubleWritable.java Same as in the previous lab assignment.

PairWritable.java Same as in the previous lab assignment.

Stemmer.java Same as in the previous lab assignment.

PublicTests.java A collection of public tests that will be run on your implementation.

BaseTests.java A superclass of PublicTests with some utility methods.

3 Submission and Evaluation

Submit your solution by Sunday, June 19, 11:59pm on the submission server for full credit. Submissions received after this date and before Monday, June 20, 8:30am will incur a penalty of 4 points out of 20. Submissions received after this second deadline will not be considered. As for the previous lab assignment, for this lab you only need to submit the `InvertedIndex.java` file, not the whole Eclipse project.

Your submission will first be run against the public tests; as they are identical to the ones provided in the PublicTests class, there is no need to use the submission server unless your submission passes these tests locally. A score out of 20 points will be given as an indication (this score is not part of the grade). When confident about your submission, you can release it to test it against the secret (release) tests. The score obtained on the release tests will be used as a grade for this lab session. You are allowed at most 3 release submissions per hour. Evaluation of your submissions will typically take several minutes.

4 Structure of `InvertedIndex.java` and What to Implement

For most of the steps to implement, you will need to read the documentation of the HBase API, from `https://hadoop.apache.org/docs/stable/api/index.html?overview-summary.html` and `https://hbase.apache.org/apidocs/index.html?overview-summary.html`. Feel free also to get documentation from other Web sources.

There are basically two main steps:

1. You need to complete the implementation of the inverted index construction, and then run this construction on the Hadoop cluster. To do so, first package your project into a JAR file by using ant: right click on the `build.xml` file within your project and select Run As Ant Build (or navigate to the directory containing this file in a terminal and type `ant`; this will produce a JAR package). Make sure the HBase table is created, and looks correct, at the end of this step.
2. You need to implement the few non-static methods at the end of `InvertedIndex.java` to retrieve information from the HBase table.

configuration is a configuration object for Hadoop; you do not need to use it directly.

NB_DOCUMENTS defines a Hadoop counter identifier that will be used for counting the number of documents.

Map is the mapper. Most of it is already written. Its fields and methods are:

- **alpha** is a regular expression for tokenization; note that only letters, not digits, are accepted within tokens this time. You do not need to use it directly.
- **stopWords** is a `HashSet` that will store stop words locally to each map task. You will need to fill it within the `retrieveStopWords` function.
- **setup** is a function called at the initialization of a mapping task. It just calls `retrieveStopWords`.
- **map** is the Map function, mostly already implemented. You need to add to this function:
  1. the filtering out of Simple English Wikipedia articles that are not from the default Wikipedia namespace; in other words, every document whose title contains a ":" (such as, "Category:Rock singers") should not be added to the index;
  2. the extraction of the article text from within the Result object;
  3. the increasing of the `NB_DOCUMENTS` counter.
- **retrieveStopWords** should load the stop word list referenced in the `filename` string (with one stop word per line in the file) and store them in an appropriate in-memory data structure. Here, `filename` is an HDFS filename. You should implement the loading from HDFS, the rest is already implemented.
- **isStopWord** returns whether a word is a stop word. This function is already implemented.

Reduce is the reducer. Most of it is already written. Its fields and methods are:

- **nb_documents** is a reference to the `NB_DOCUMENTS` counter, initialized within the `setup` function. You do not need to use it directly.
- **setup** is a function called at the initialization of a reducing task. It initializes `nb_documents`.
- **reduce** is the Reduce function, mostly already implemented. You need to add to this function the production of a `KeyValue` object that contains row key, family, qualifier, and value corresponding to an inverted index entry.

buildInvertedIndex is the function that launches the MapReduce job. It is mostly implemented, but you need to add to the `Scan` object the columns that need to be retrieved, and you need to set

an `outputPath` that points to an HDFS location to store a file containing the inverted index object before it is loaded into HBase.

DocumentWeight is already implemented, and mostly as in the first lab assignment.

main simply calls `buildInvertedIndex`, providing it with the names of the tables to read from and to write to.

getLogin is supposed to return a constant string formed of your login; you need to implement this.

table is a reference to the table where you stored the inverted index, it is initialized within the default constructor.

InvertedIndex() , the default constructor, is supposed to initialize the `table` fields; you need to implement this.

getScore is supposed to load from the inverted index stored in HBase (once this inverted index has been produced) the score of a token in a document (or -1 if the token does not occur in the document); you need to implement this.

getNumberColumns is supposed to return the number of columns of a given row of the HBase table (alternatively, the size of the posting list for a given token); you need to implement this.