Lab exercises for theme NoSQL

Aim

After completing this lab you should have the basic knowledge on programming environment and the programming techniques of Hadoop. In the first part of the lab you will go through the tutorial on the WordCount example at Apache website. After the first part of the lab you will be familiar with the programming environment of Hadoop system. In the second part of the lab you will solve the XPath query answering problem using the Hadoop system, by first generating the index structure of the given XML data and then writing the XPath query in Java over the Hadoop system.

1. WordCount v1.0

The first task is to go through the Hadoop tutorial that can be found at:
http://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0

The Hadoop server at NSC is:

`heffa.nsc.liu.se` (denoted later as `heffa`)

You will be able to log in `heffa` using `SSH` from any lab computer or your personal computer. Use your student account at NSC to log in (NOTE: Please apply for an NSC account as soon as possible since manual steps are involved). You will be working with the Linux system. After you are logged in, type

```
whereis hadoop
```

to find out where the hadoop executable and the libs are installed. You will see the following message:

hadoop: /usr/bin/hadoop /usr/lib/hadoop /etc/hadoop /usr/share/man/man1/hadoop.1.gz

`/usr/bin/Hadoop` is the path for the executable and you should always give this path when calling the hadoop command.
`/usr/bin/hadoop` replaces `bin/hadoop` in the tutorial
`hdfs` replaces `bin/hdfs`

To copy files from your local machine to `heffa` you need to use commands like `scp`, `rsync`, etc. Here is an example with `scp`:

```
scp filename username@heffa.nsc.liu.se:~
```

(will move `filename` to your home folder on `heffa`)

Note that the command above will move the file on `heffa` but NOT on the `hdfs`. To do so you need to use `hdfs dfs` command as below:

```
hdfs dfs -put -f filename ./input
```

(assuming your current folder is where `filename` is, the command will move `filename` in the `input` folder (which should be created in advance))
The input and output files have to reside under `hdfs` but not your code!

Now you should be able to follow the tutorial and run the WordCount program in Hadoop. (hint: the output data directory should not exist before you run the program. Otherwise a “directory already exists” error message will be returned).

2. XPath query answering using Hadoop

In this part you will learn how to write your own application using Hadoop. The task consists of two steps.

2.1 Parsing the XML file and storing the index structure in text files.

We begin with building the path-based mapping together with the dewey encoding for the given XML data. The XML data can be downloaded at:
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/BIOMD0000000009.xml

This paper describes how to use dewey encoding (**very detailed example is available at the end of the file**):
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/StoringAndQueryingOrderedXMLUsingARelationalDatabaseSystem.pdf

In this exercise we will use the dewey encoding to store the positions of the nodes, (i.e. every node in the XML tree will be assigned a unique dewey_pid). For each *unique* path of the XML tree, you need to generate three text files: one for the tags (tag file), one for the texts (text file), and one for the attributes and values (attribute file). Tag files consist of the pairs of the tag names together with its dewey_pid with the format:

```
      ...
   1.1.3.4  tagname1
   1.1.3.5  tagname2
      ...
```

Text file has the similar struture, with pairs of text values together with the dewey_pid.
Attribute file consists of the triple with the format of

```
      ...
   1.1.3.4  attributename1 attributevalue1
   1.1.3.5  attributename2 attributevalue2
      ...
```

To generate the files you will use the standard XML event-based parser SAX (read about it, if you are not familiar with it already). Two Java files are prepared for you to be completed. The files can be downloaded under:
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/XMLCounter.java
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/ReadXMLFile.java

Your task is to add the content to the functions in XMLCounter.java such as startElement, endElement, etc. In order to generate the paths and the dewey code, you might need the data structures such as Stack and Vector. To complete this task you may use any Java IDE such as Eclipse.

2.2 XPath query evaluation

The purpose of this part is to use the MapReduce programming model to evaluate the following XPath query. Consider the following XPath query over the given XML data:

//reaction[listOfProducts/speciesReference[contains(@species, "P_KK")]]/listOfReactants/speciesReference/@species

Intuitively, the query asks for the reactant species of the reactions which consist of a product species contain the substring 'P_KK'.

The query can be evaluated by first retrieving the attribute file (file01) with the path:
*/reaction/ListOfProducts/speciesReference
and another attribute file (file02) with the path:
*/reaction/ListOfReactants/speciesReference

*To simplify the task, you do not need to implement the pattern matching steps of the paths. From the example XML data we know there is only one path contains "reaction/ListOfProducts/speciesReference" (resp."reaction/*ListOfReactants*/speciesReference"). Thus from all the files you have generated in the last task, there is one attribute file (file01) for the first path and one (file02) for the second path. You need to remember the file names and give them as input parameters for the execution of the program.*

Note that the files contain all the products (resp. reactants) from all the reactions. Now we need to conduct a join operation on both data sets by grouping the products and reactants from the same reaction. Therefore, the join attribute is the dewey_pid of the path */reaction. Given a dewey_pid (id) from file01 or file02, we can easily obtain the dewey_pid of */reaction by removing the last two digits from id. For instance, if the dewey_pid from file01 is
1.1.3.4
then the dewey_pid of */reaction would be
1.1

The join operation is realized by the following two map tasks:

Map task 1 (works on the first part of the query *[listOfProducts/speciesReference[contains(@species,"P_KK*")]]): Take a triple from file01, check whether the attribute name is "species" and the attribute value contains the string "P_KK". If this is true, retrieve the prefix of the current dewey_pid and send it as the key to the reduce task.

Map task 2 (works on the second part */listOfReactants/speciesReference/@species*): Similar to Map task 1.

The function in the reduce task is then straightforward.

We prepared the necessary Java files for you to fill up the contents. The files are available at:
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/TextPair.java
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinMain.java
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinMapper1.java
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinMapper2.java
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinReducer.java

You will need all the files to complete the task. From the above files, TextPair.java and JoinMain.java are complete and you should not change them. Your task is to complete the other three files.

There are two Mappers and each one of them sends the intermediate <key, value> pairs to the reduce task, where the pairs with the same key are sent to the same reduce task. At the reducer side, there should be mechanism to identify for a received data pair, which Mapper does it come from. The solution is to tag the pairs in the Mapper side with some distinct number.
This kind of join operation is known as Reduce-side Join.

Compiling & running:

Under `/usr/lib/hadoop` and `/usr/lib/hadoop-mapreduce` you will find the jar files serving as lib files of hadoop system. Main jar files are `hadoop-common.jar` in `/usr/lib/hadoop` and `hadoop-mapreduce-client-core.jar` in latter.

To compile source code, you could use the following command:

```
javac -cp /usr/lib/hadoop/hadoop-common.jar:/usr/lib/hadoop-
mapreduce/hadoop-mapreduce-client-core.jar -d YOUR_CLASS_PATH *.java
```

or compile as WordCount program in the tutorial assuming environment variables are set.

```
/usr/bin/hadoop com.sun.tools.javac.Main *.java
```

Then create a jar file and execute it similarly to what you did in the Word Count example. Look at the JoinMain.java for the number and sequence of the program arguments.

Console output & Debugging:

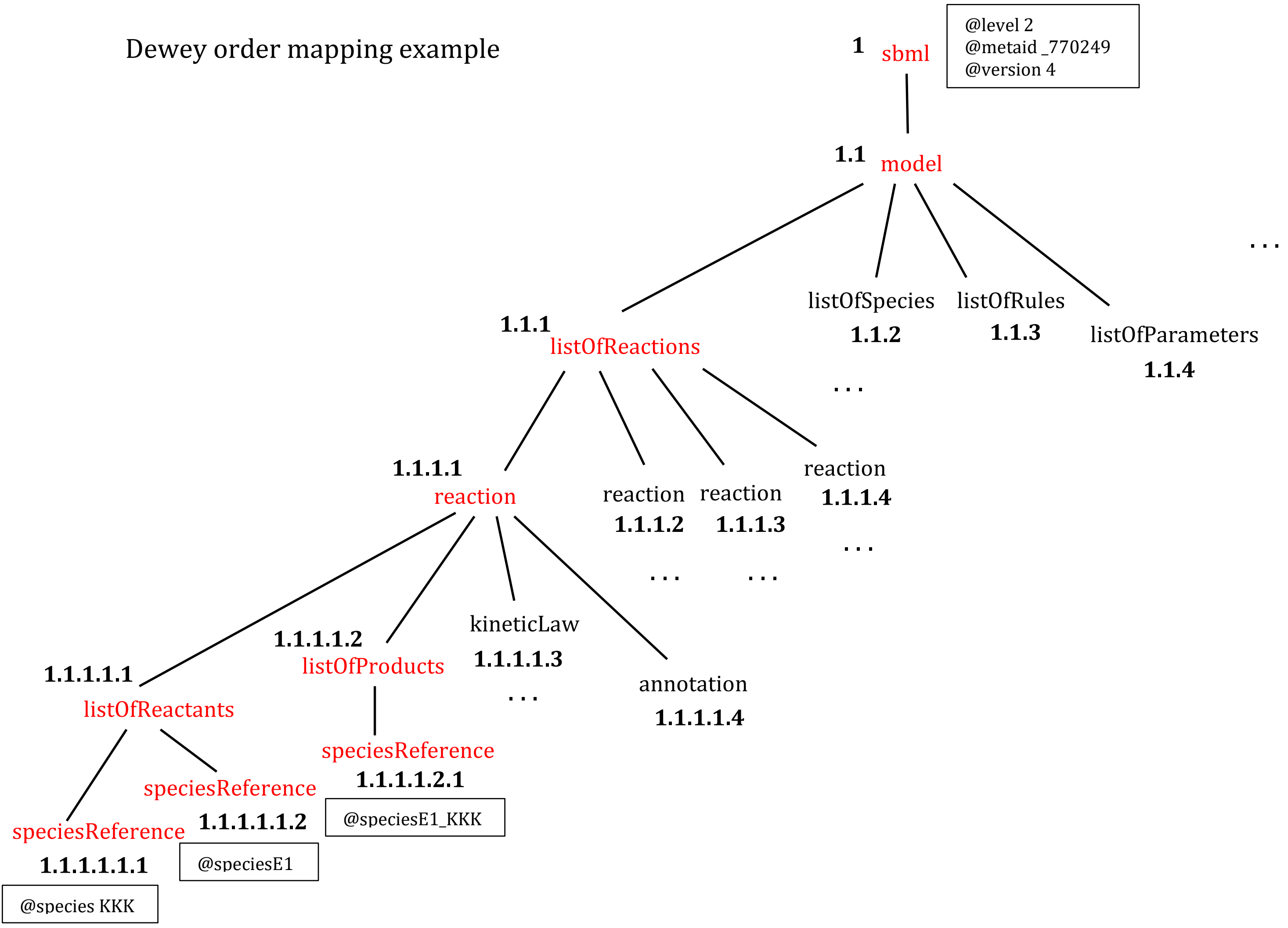
To debug your code you can use regular System.out statements. An example how to access their output in Hadoop environment can be found here:
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/debug_example_wordcount_1.0.zip
You can run the following using your application ID:

```
yarn logs -applicationId application_1506513566622_0010
```

Hand in:

To demonstrate the lab, please e-mail a lab report containing all code you have written for both part 2.1 and part 2.2 and also include a file of what it displays when it's run.

Dewey order mapping example



The content of the attribute file for the *sbml* path:
1 level 2
1 metaid _770249
1 version 4

The content of the tag file for the *sbml* path:
1 sbml

The content of the text file for the *sbml* path:
(empty since thete is no text for the element with this path)

The content of the attribute file for the *sbml/model/listOfReactions/reaction/listOfReactants/speciesReference* path:
1.1.1.1.1.1 species KKK
1.1.1.1.1.2 species E1
... continue with the other attributes with this path

The content of the tag file for the *sbml/model/listOfReactions/reaction/listOfReactants/speciesReference* path:
1.1.1.1.1.1 speciesReference
1.1.1.1.1.2 speciesReference
... continue with the other elements with this path

The content of the text file for the *sbml/model/listOfReactions/reaction/listOfReactants/speciesReference* path:
(empty since thete is no text for the elements with this path)

The content of the attribute file for the *sbml/model/listOfReactions/reaction/listOfProducts/speciesReference* path:
1.1.1.1.2.1 species E1_KKK
... continue with the other attributes with this path

The content of the tag file for the *sbml/model/listOfReactions/reaction/listOfProducts/speciesReference* path:
1.1.1.1.2.1 speciesReference
... continue with the other elements with this path

The content of the text file for the *sbml/model/listOfReactions/reaction/listOfProducts/speciesReference* path:
(empty since thete is no text for the elements with this path)