TDDD43 - Lab Exercises for Theme NoSQL

***Notice**: Please make sure you have read the whole lab compendium before you start to work on the server from NSC.*

Description and Aim

In this lab you will work on the Sigma[1] set up which is a HPC cluster from the National Supercomputer Centre (NSC). You are supposed to solve an XPath query answering problem using the Hadoop system on Sigma, by first generating the index structure of the given XML data (section 1.1) and then writing an XPath query evaluation program in Java following Apache Hadoop 2.7.0[2] (section 1.2). Finally you need to compile your code for the XPath query evaluation and run the program, on Sigma (section 2). After completing this lab, you should have basic knowledge on the programming environment and the programming techniques of Hadoop.

1. XPath query answering

The task consists of two parts. In the first part, you will learn how to parse the XML file. In the second part, you will learn how to write a Java program using MapReduce based on Hadoop.

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

(You can program locally on your own machines, then upload the outputs which are input for XPath query evaluation to Sigma.)

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

The following paper describes how to use dewey encoding (a detailed example is available at the end of this section shown in Figure 1):

https://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 their dewey_pids with the format as follows.

...
1.1.3.4 tagname1
1.1.3.5 tagname2
...

Text file has a similar structure, with pairs of text values together with the dewey_pids.
Attribute file consists of the triple with the format as follows.

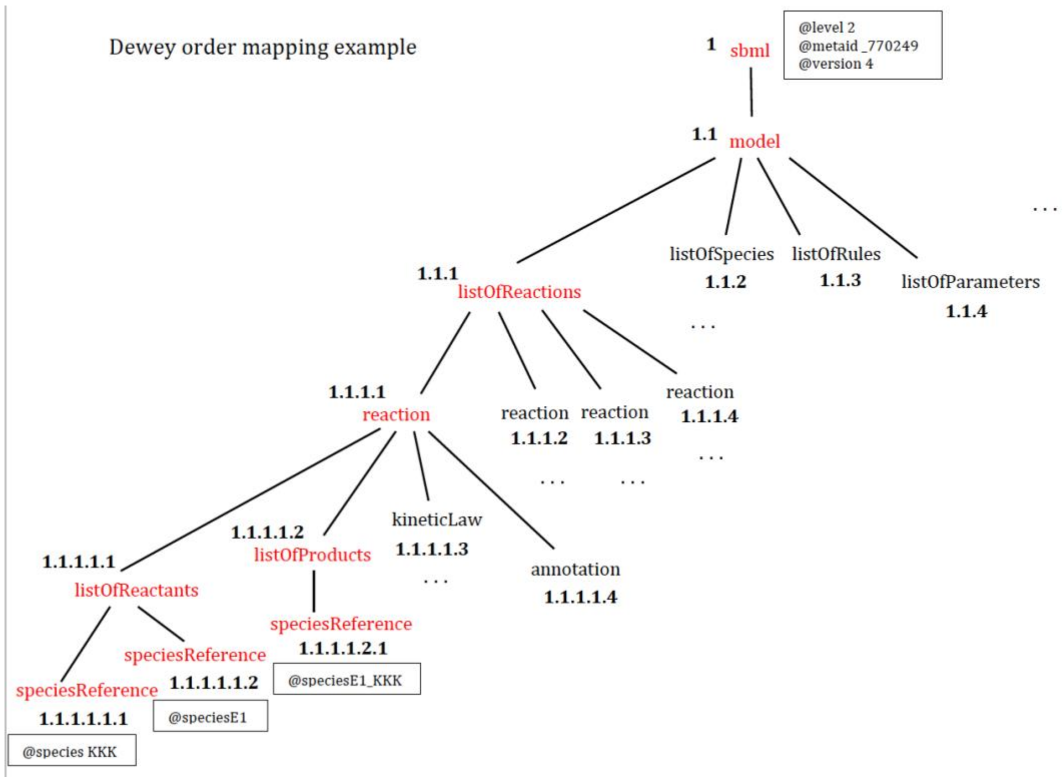
...
1.1.3.4 attributename1 attributevalue1
1.1.3.5 attributename2 attributevalue2
…

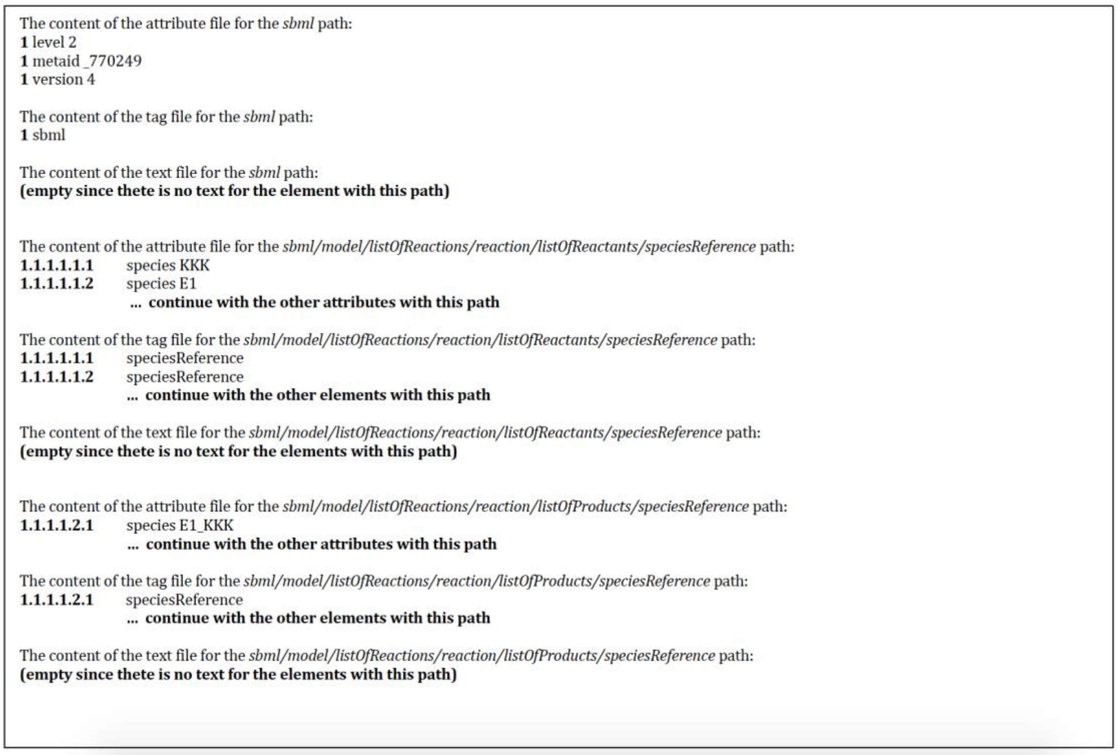
To generate the files, you will use the standard XML event-based parser SAX (read about it, if you are not familiar with it already). Two Java files are prepared for you to be completed. The files can be downloaded at:

http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/XMLCounter.java
http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/ReadXMLFile.java

[1] Sigma: https://www.nsc.liu.se/systems/sigma/
[2] Apache Hadoop 2.7.0: https://hadoop.apache.org/docs/r2.7.0/

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




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)

Figure 1: Dewey order mapping Example

1.2 XPath query evaluation

(You can also start to do section 2.1 and 2.2 to run WordCount program, then get back to this section.)

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 containing:

"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:

In map task 1 (works on the first part of the query *[listOfProducts/speciesReference[contains(@species,"P_KK")]]*), you need to 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.

In map task 2 (works on the second part */listOfReactants/speciesReference/@species*), what you need to do is 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.

2. Running Java Hadoop Program on Sigma

2.1 Working on Sigma

The Sigma server is available at `sigma.nsc.liu.se` (log in using your NSC accounts). You can use ssh forwarding connection or Thinlinc connection to log in Sigma. It's also fine to use regular ssh connection without forwarding option, in this case, you have to program locally and use scp command to upload your code to Sigma.

To log in Sigma:

- Thinlinc server is available same as `sigma.nsc.liu.se`. In this way, you can get a graphical environment on Sigma and given that you work directly on Sigma there is no need to use `ssh` or `scp`. When you use Thinlinc, don't forget to uncheck the full screen mode so that you can share the screen with your lab partner or lab assistant when you ask questions. **Note**: Please remember to log out when done working on the labs so that Sigma does not keep open Thinlinc sessions. Also, each pair of students, please uses one Thinlinc connection during lab sessions, due to the limited number of Thinlinc licenses on Sigma.
- `ssh -X username@sigma.nsc.liu.se` where username is your NSC username (not the LiU one), `-X` indicates forwarding function of `ssh` which is used for running graphics applications remotely.
- `[username@sigma ~] $ exit` is used to logout Sigma. If it is hung on, please use ctrl-c to terminate the connection.
- `[username@sigma ~] $ emacs &` You can use Emacs for coding by running `emacs &` in the terminal after you connect to Sigma or program locally on your machine, then use scp command to copy you files to Sigma.
- `scp LOCAL_FILE_PATH username@sigma.nsc.liu.se:Documents` is used for uploading files from your local machine to Sigma. (**Note**: you are supposed to run scp command before you log in Sigma when you want to upload files to Sigma.)

We will use a non-interactive way to run the program on Sigma. So each time when you want to run your java program, you need to submit your batch job. After that, the job will enter the scheduling queue, where it may have to wait a while until nodes are available to tun the job. We use `sbatch` command to submit the batch job and `squeue` command to monitor your submitted job. You may also use `scancel` command to cancel a job.

- `[username@sigma ~]$ sbatch run.q`
- `[username@sigma ~]$ squeue -u username`
- `[username@sigma ~]$ scancel JOBID`

The Sigma uses `Slurm` for scheduling. Once you submit a job, the job will be assigned an ID. After the job is finished, you will see a `slurm-ID.out` file returned, which includes the output information of the job script.

We provide a script named `compile.sh` to compile your java code and a script named `run.q` to run the program. Please read the scripts carefully before you change it. For compiling your java code, you don't have to submit the job for scheduling. You can simply run the script as follows.

- `[username@sigma ~]$ ./compile.sh`

Notice: NSC reserves 6 compute nodes for the two lab sessions in TDDD43 which means other jobs on Sigma will not use these nodes during our lab sessions. To use the reservations, you can use following commands.

- `[username@sigma ~]$ listreservations`

```
Reservations available to user:username / project(s):liu-compute-2020-38
RESERVATION_NAME from START_TIME to END_TIME (project: liu-compute-2020-38)
RESERVATION_NAME from START_TIME to END_TIME (project: liu-compute-2020-38)
devel  from NOW to INF  (everyone)
```

Each time when you use sbatch command to submit a job to the cluster, please specify the project parameter with `-A` option and `--reservation` option as follows:

- `[username@sigma ~]$ sbatch -A liu-compute-2020-38 --reservation=devel run.q`

2.2 Running Word Count Example on Hadoop

Step 1: Log in to Sigma server.

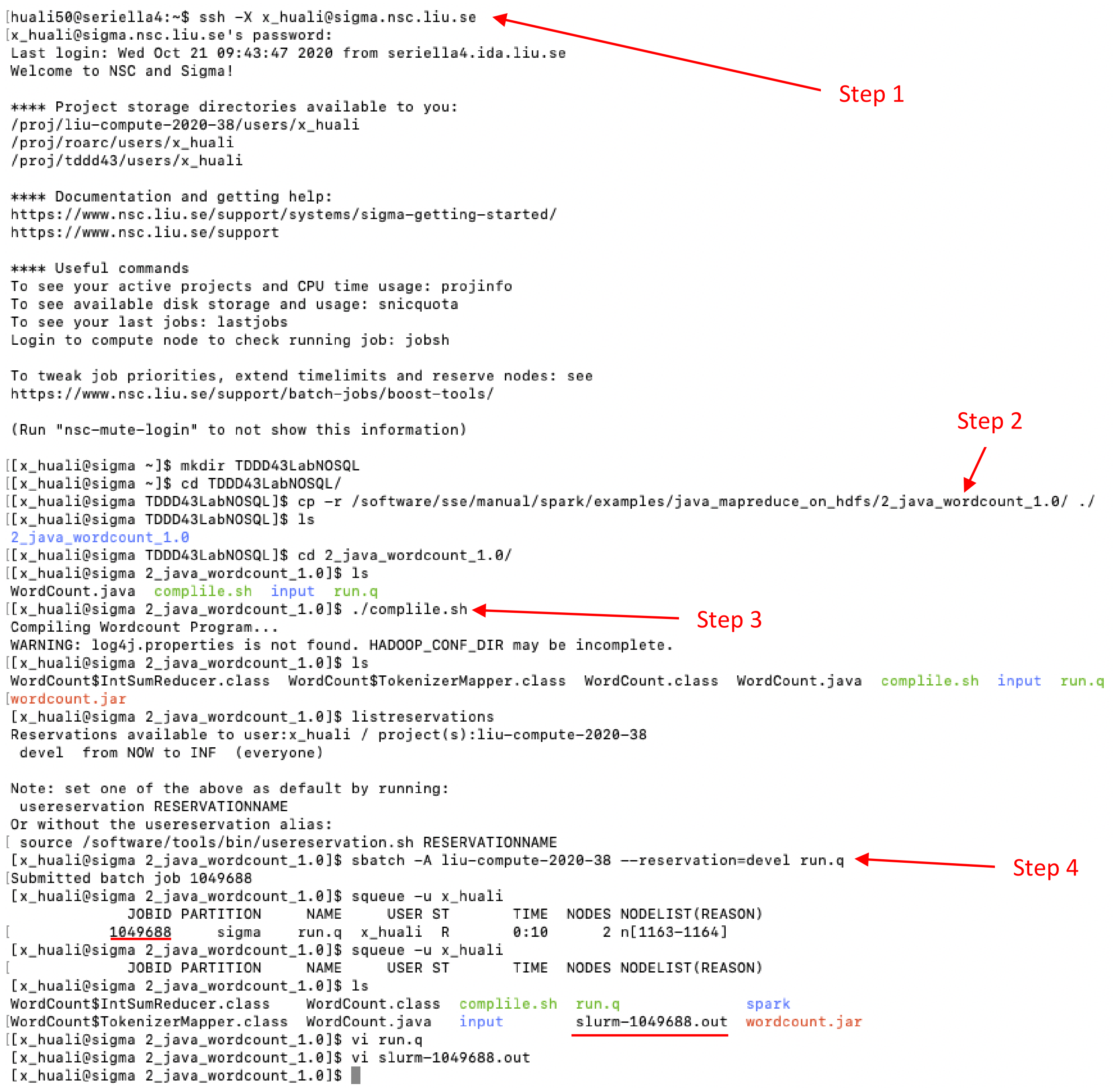
Step 2: There is an example for how to run Word Count program with Hadoop on Sigma. The path for the example is:

/software/sse/manual/spark/examples/java_mapreduce_on_hdfs/2_java_wordcount_1.0/

You can copy this folder to your home folder by using `cp` command.

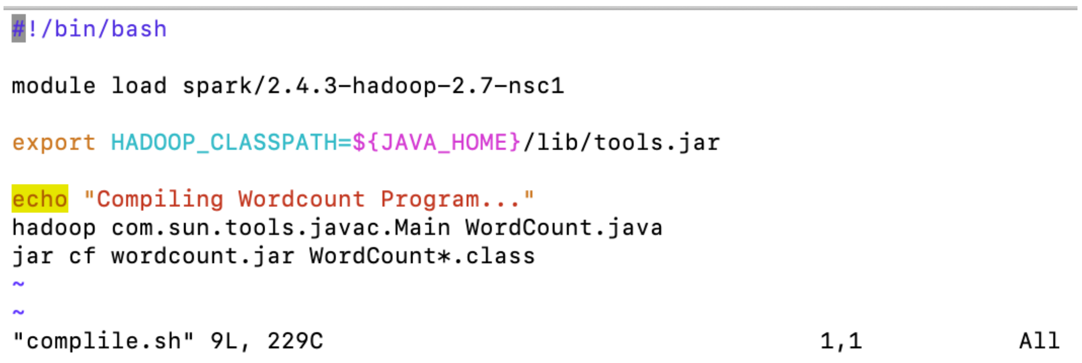
Step 3: Before you run the program on Hadoop, you need to first compile the code by running the script `compile.sh`. The script is used to Compile `WordCount.java` and create a jar as shown in Figure 3.

Step 4: After the jar file is created, you can use `sbatch` to submit the job. Then you can use `squeue -u USER_NAME` to check your job. Once the job is finished, you can find a returned file named `slurm-ID.out`.



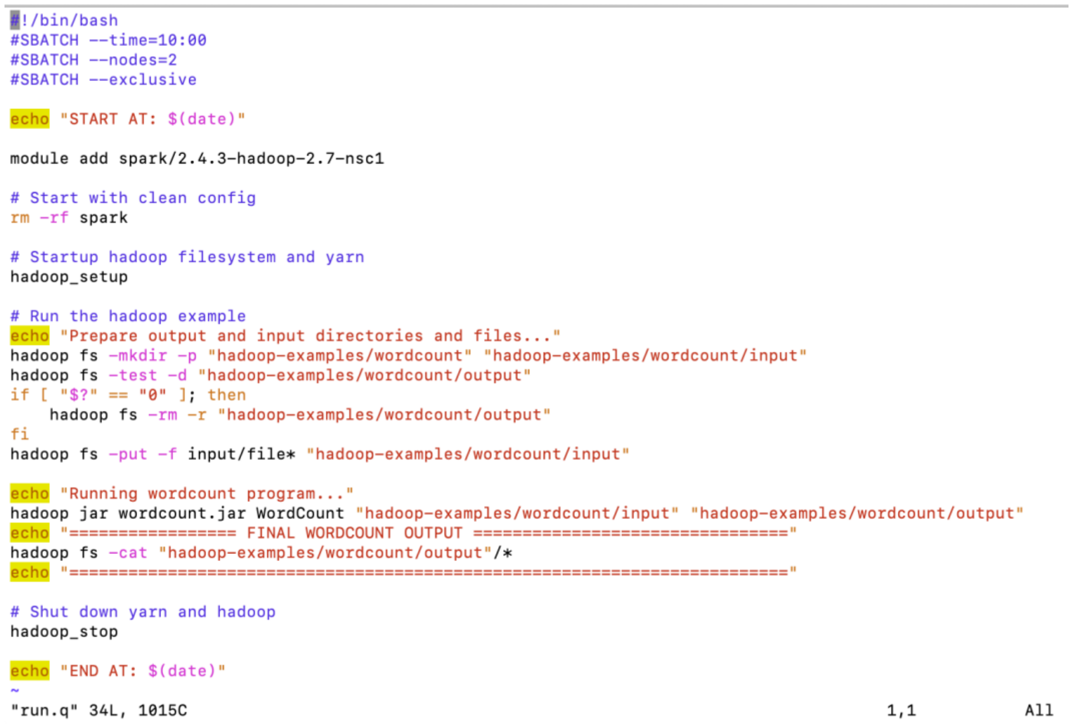
```
huali50@seriella4:~$ ssh -X x_huali@sigma.nsc.liu.se
x_huali@sigma.nsc.liu.se's password:
Last login: Wed Oct 21 09:43:47 2020 from seriella4.ida.liu.se
Welcome to NSC and Sigma!

**** Project storage directories available to you:
/proj/liu-compute-2020-38/users/x_huali
/proj/roarc/users/x_huali
/proj/tddd43/users/x_huali

**** Documentation and getting help:
https://www.nsc.liu.se/support/systems/sigma-getting-started/
https://www.nsc.liu.se/support

**** Useful commands
To see your active projects and CPU time usage: projinfo
To see available disk storage and usage: snicquota
To see your last jobs: lastjobs
Login to compute node to check running job: jobsh

To tweak job priorities, extend timelimits and reserve nodes: see
https://www.nsc.liu.se/support/batch-jobs/boost-tools/

(Run "nsc-mute-login" to not show this information)

[x_huali@sigma ~]$ mkdir TDDD43LabNOSQL
[x_huali@sigma ~]$ cd TDDD43LabNOSQL/
[x_huali@sigma TDDD43LabNOSQL]$ cp -r /software/sse/manual/spark/examples/java_mapreduce_on_hdfs/2_java_wordcount_1.0/ ./
[x_huali@sigma TDDD43LabNOSQL]$ ls
2_java_wordcount_1.0
[x_huali@sigma TDDD43LabNOSQL]$ cd 2_java_wordcount_1.0/
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount.java  complile.sh  input  run.q
[x_huali@sigma 2_java_wordcount_1.0]$ ./complile.sh
Compiling Wordcount Program...
WARNING: log4j.properties is not found. HADOOP_CONF_DIR may be incomplete.
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount$IntSumReducer.class  WordCount$TokenizerMapper.class  WordCount.class  WordCount.java  complile.sh  input  run.q
wordcount.jar
[x_huali@sigma 2_java_wordcount_1.0]$ listreservations
Reservations available to user:x_huali / project(s):liu-compute-2020-38
 devel  from NOW to INF  (everyone)

Note: set one of the above as default by running:
 usereservation RESERVATIONNAME
Or without the usereservation alias:
 source /software/tools/bin/usereservation.sh RESERVATIONNAME
[x_huali@sigma 2_java_wordcount_1.0]$ sbatch -A liu-compute-2020-38 --reservation=devel run.q
Submitted batch job 1049688
[x_huali@sigma 2_java_wordcount_1.0]$ squeue -u x_huali
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1049688     sigma    run.q  x_huali  R       0:10      2 n[1163-1164]
[x_huali@sigma 2_java_wordcount_1.0]$ squeue -u x_huali
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[x_huali@sigma 2_java_wordcount_1.0]$ ls
WordCount$IntSumReducer.class    WordCount.class  complile.sh  run.q                spark
WordCount$TokenizerMapper.class  WordCount.java   input        slurm-1049688.out    wordcount.jar
[x_huali@sigma 2_java_wordcount_1.0]$ vi run.q
[x_huali@sigma 2_java_wordcount_1.0]$ vi slurm-1049688.out
[x_huali@sigma 2_java_wordcount_1.0]$ 
```

Figure 2: Steps to run word count program



```
#!/bin/bash

module load spark/2.4.3-hadoop-2.7-nsc1

export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

echo "Compiling Wordcount Program..."
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wordcount.jar WordCount*.class
~
~
"complile.sh" 9L, 229C                                  1,1           All
```

Figure 3: compile.sh for word count program



```
#!/bin/bash
#SBATCH --time=10:00
#SBATCH --nodes=2
#SBATCH --exclusive

echo "START AT: $(date)"

module add spark/2.4.3-hadoop-2.7-nsc1

# Start with clean config
rm -rf spark

# Startup hadoop filesystem and yarn
hadoop_setup

# Run the hadoop example
echo "Prepare output and input directories and files..."
hadoop fs -mkdir -p "hadoop-examples/wordcount" "hadoop-examples/wordcount/input"
hadoop fs -test -d "hadoop-examples/wordcount/output"
if [ "$?" == "0" ]; then
    hadoop fs -rm -r "hadoop-examples/wordcount/output"
fi
hadoop fs -put -f input/file* "hadoop-examples/wordcount/input"

echo "Running wordcount program..."
hadoop jar wordcount.jar WordCount "hadoop-examples/wordcount/input" "hadoop-examples/wordcount/output"
echo "================= FINAL WORDCOUNT OUTPUT ================================"
hadoop fs -cat "hadoop-examples/wordcount/output"/*
echo "========================================================================="

# Shut down yarn and hadoop
hadoop_stop

echo "END AT: $(date)"
~
"run.q" 34L, 1015C                                                      1,1           All
```

Figure 4: run.q script for word count program

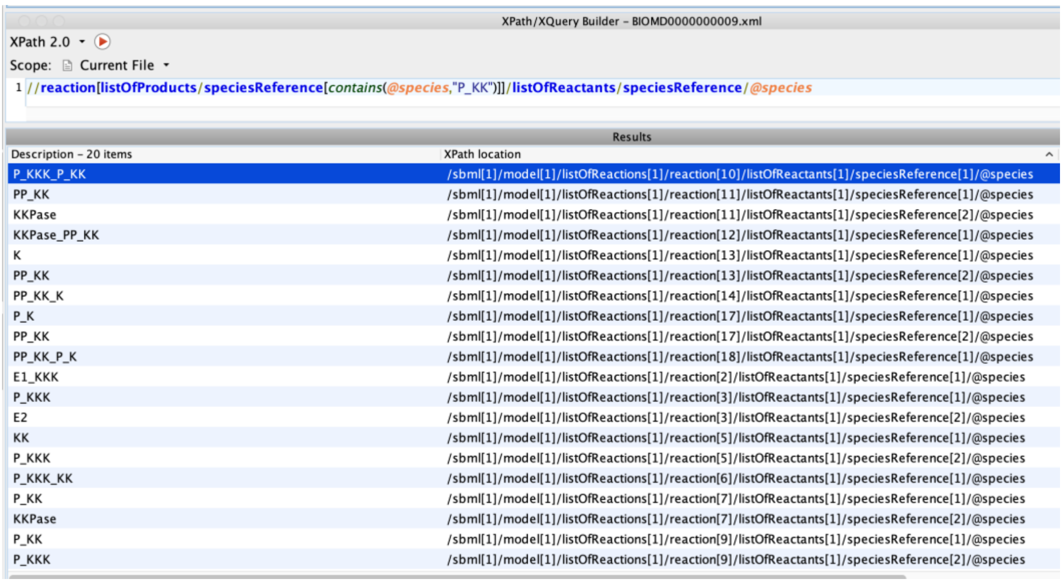
In *run.q* as shown in Figure 4, there are a number of commands that are used to interact with HDFS.

hadoop fs -mkdir <FOLDER_NAME>	-make a folder on HDFS
hadoop fs -mkdir -p <FOLDER_NAME> <FOLDER_NAME>	-make multiple folders
hadoop fs -test -d <FOLDER_NAME>	-if the path is a directory, return 0
hadoop fs -rm -r <FOLDER_NAME>	-deletes the directory and any content under it recursively
hadoop fs -put <localsrc> ... <dst>	-copy single src, or multiple srcs from local to HDFS
hadoop fs -cat <FOLDER_ON_HDFS> [local]	-copy HDFS path to stdout

2.3 Running XPath evaluation program on Hadoop

After you have achieved correct results for section 1.1 and have written correct code for section 1.2, you can prepare to run your program on Hadoop. To compile the Java code and to run the program are similar to step 3 and step 4 in section 2.2, respectively. Indeed, you have to change some details in the two scripts, *compile.sh* and *run.q*.

Figure 5 shows the XPath query result in Oxygen, you are supposed to obtain the same 20 records in your program.


XPath/XQuery Builder – BIOMD0000000009.xml

XPath 2.0

Scope: Current File

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

Results

Description – 20 items	XPath location
P_KKK_P_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[10]/listOfReactants[1]/speciesReference[1]/@species
PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[11]/listOfReactants[1]/speciesReference[1]/@species
KKPase	/sbml[1]/model[1]/listOfReactions[1]/reaction[11]/listOfReactants[1]/speciesReference[2]/@species
KKPase_PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[12]/listOfReactants[1]/speciesReference[1]/@species
K	/sbml[1]/model[1]/listOfReactions[1]/reaction[13]/listOfReactants[1]/speciesReference[1]/@species
PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[13]/listOfReactants[1]/speciesReference[2]/@species
PP_KK_K	/sbml[1]/model[1]/listOfReactions[1]/reaction[14]/listOfReactants[1]/speciesReference[1]/@species
P_K	/sbml[1]/model[1]/listOfReactions[1]/reaction[17]/listOfReactants[1]/speciesReference[1]/@species
PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[17]/listOfReactants[1]/speciesReference[2]/@species
PP_KK_P_K	/sbml[1]/model[1]/listOfReactions[1]/reaction[18]/listOfReactants[1]/speciesReference[1]/@species
E1_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[2]/listOfReactants[1]/speciesReference[1]/@species
P_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[3]/listOfReactants[1]/speciesReference[1]/@species
E2	/sbml[1]/model[1]/listOfReactions[1]/reaction[3]/listOfReactants[1]/speciesReference[2]/@species
KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[5]/listOfReactants[1]/speciesReference[1]/@species
P_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[5]/listOfReactants[1]/speciesReference[2]/@species
P_KKK_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[6]/listOfReactants[1]/speciesReference[1]/@species
P_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[7]/listOfReactants[1]/speciesReference[1]/@species
KKPase	/sbml[1]/model[1]/listOfReactions[1]/reaction[7]/listOfReactants[1]/speciesReference[2]/@species
P_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[9]/listOfReactants[1]/speciesReference[1]/@species
P_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[9]/listOfReactants[1]/speciesReference[2]/@species

Figure 5: XPath query result in Oxygen

Hand in:

To demonstrate the lab, please e-mail a lab report containing all code you have written for both section 1.1 and section 1.2, and also include a file to show the result of your program.