# EE 382N: Distributed Systems
# Homework 3

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)
TA: Asad Malik (email:asadsm@utexas.edu )

**Deadline: Nov. 3, 2017**

This homework contains a theory part (Q1-Q2) and a programming part (Q3). The theory part should be written or typed on a paper and submitted at the beginning of the class. The source code must be uploaded through Canvas before the end of the due date (i.e., 11:59pm on Nov. 3). The assignment should be done in teams of two. In addition, you should not use package for encapsulation. Please zip and name the source code as [EID1_EID2].zip.

1. **(10 pts)** Consider the global predicate $B = (x_1 < x_2) \wedge (x_3 > 0)$ where $x_i$ is local variable on process $P_i$. Your task is to design an algorithm to detect this predicate in an offline fashion. In an offline algorithm, a trace is generated for each process. After the computation is finished all the traces are sent to a checker process which is responsible for detecting the predicate. You need to show what information should be kept as part of the trace. Also, give the algorithm for the checker process. What is the time complexity of the checker process if a process has at most $m$ state intervals?

2. **(10 pts)** The algorithm proposed by Chandy and Lamport (in Figure 9.3 of the textbook) works correctly when channels are FIFO. However, even if channels are not FIFO, the algorithm would work correctly if we can guarantee the following condition on all channels:

   **(C1)** Any application message sent before the marker is delivered before the marker and any application message sent after the marker is delivered after the marker.

   Suppose that the underlying channels do not satisfy FIFO. How will you implement condition (C1). Your implementation should not inhibit delivery of a message unless it violates (C1).

3. **(80 pts)** In this problem, you will implement a counting indexer using Hadoop, a programming model and software framework for developing applications that concurrently process large scale data (Big Data) on distributed systems. The indexer you develop has to identify the files and calculate the occurrences of every word that appear in the given set of files. The data set for this problem is the novel *Pride and Prejudice*. A collection of files (one per chapter) of all the text is provided in the attached zip file for the assignment. **Your implementation must be able to run in Single-Node mode using Hadoop 2.6.5**

   **Output Format:** Your output should follow the format (and only this format) of:

   ```
   word1
   <file-name1, occurrence-frequency1>
   ```

```
<file-name2, occurrence-frequency2>
⋮

word2
<file-name3, occurrence-frequency3>
<file-name4, occurrence-frequency4>
⋮
```

**Ex.** If we run the text indexer on the full text of the novel, we find that *Darcy* occurs 41 times in chap18, 21 times in chap10 and chap 16, 18 times in chap 33, and so on. Your output should be as follows:

```
darcy
<chap18, 41>
<chap10, 21>
<chap16, 21>
<chap33, 18>
⋮
```

Note that, your outputs should be sorted base on the occurrence-frequency. If two chapters have the same occurrence-frequency, then you should print the earlier chapter first.

For simplifying this assignment and obtaining the same results, you should 1) convert every character into lower-case, 2) mask non-alphabetic characters by white-space; i.e., any character other than `a-z` should be replaced by a single-space character.

Before you start working on this assignment, you should read the Tutorial on Hadoop webpage for developing a Hadoop applicatoin. In this assignment, we will use Hadoop 2.6.5. The following steps will guide you through the setup of Hadoop 2.6.5 on a Linux machine.

**Setup Hadoop in Single Node Mode (which is called Pseudo-Distributed Mode in r1.1.2)**:

(a) Download Hadoop 2.6.5 (hadoop-2.6.5.tar.gz) by clicking "Download a release now" in the webpage:
http://hadoop.apache.org/releases.html#Download [Download the binary for 2.6.5 from the link]

(b) Untar the downloaded tarball into a directory.

(c) Add the following lines to your .bashrc:

```
export HADOOP_HOME=/path/to/your/untarred/hadoop-2.6.5
export HADOOP_MAPRED_HOME=$HADOOP_HOME  # home to MapReduce module
export HADOOP_COMMON_HOME=$HADOOP_HOME  # home to common module
export HADOOP_HDFS_HOME=$HADOOP_HOME    # home to HDFS module
export YARN_HOME=$HADOOP_HOME           # home to Yarn module
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop # path to configurations
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin # add Hadoop binaries to PATH
```

After adding those lines into your .bashrc, you can use the command "source .bashrc" to make the changes happen, or restart/re-login your terminal/machine.

(d) Create storages for namenode and datanode. In this example, the folders for the storages are located in `$HADOOP_HOME/hdfs`; however, you can put them in any directory you want.

```
mkdir -p $HADOOP_HOME/hdfs/namenode
mkdir -p $HADOOP_HOME/hdfs/datanode
```

(e) Setup the configuration files, which are located in `$HADOOP_HOME/etc/hadoop`, for Hadoop and Yarn. Put the following settings between <configuration> and </configuration> in each file.

i. yarn-site.xml: we add the default shuffle function into the system.

```
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

ii. core-site.xml: create a single node on the localhost. In this example, we use port 9000 for hadoop. You can use any other available port. Note that, you must ensure that the port you want to use is available.

```
<property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
</property>
```

iii. hdfs-site.xml: specify the storage for HDFS. Note that, `$HADOOP_HOME` should be replaced by **the absolute path**.

```
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:$HADOOP_HOME/hdfs/namenode</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:$HADOOP_HOME/hdfs/datanode</value>
</property>
```

iv. mapred-site.xml: this file does not exist at first, so use "`cp mapred-site.xml.template mapred-site.xml`" to create an initial configuration file, then put the setting into the file.

```
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
```

v. hadoop-env.sh and yarn-env.sh(optional): replace "`export JAVA_HOME=${JAVA_HOME}`" with "`export JAVA_HOME=/the/path/to/your/jvm/home`" if your Hadoop cannot find Java.

(f) Before we start Hadoop services in the system, we need to format the storage for namenode.

```
hdfs namenode -format
```

If you see "`INFO util.ExitUtil: Exiting with status 0`" in the last few lines, then you are good. If the returned status is 1, you can check the error message on the screen and fix the problem.

(g) Start the Hadoop daemons one by one using the following commands:

```
hadoop-daemon.sh start namenode
hadoop-daemon.sh start datanode
hadoop-daemon.sh start secondarynamenode
yarn-daemon.sh start resourcemanager
yarn-daemon.sh start nodemanager
mr-jobhistory-daemon.sh start historyserver
```

(h) Check everything is working

i. Check the log files in `$HADOOP_HOME/logs`.

ii. The following webpage should be accessible:

```
http://localhost:50070/ for NameNode
http://localhost:8088/cluster for ResourceManager
http://localhost:19888/jobhistory for Job History Server
```

If you are using an ECE machine, you might not be able to access the webpages because of firewall.

iii. Use "`jps`" command to list the Java daemons in the background. If all the processes are started successfully, you should see something similar to the following example:

```
22837 NodeManager
23465 Jps
23023 JobHistoryServer
22540 ResourceManager
16804 NameNode
17056 SecondaryNameNode
16927 DataNode
```

Every line starts with a process ID and then followed by the process name.

(i) Run a Hadoop example.

```
mkdir input  # create a local folder
vim input/file  # create and edit a file. Type down some words in the file.
hdfs dfs -copyFromLocal input/ /input # copy the local directory to HDFS
hdfs dfs -ls /input # list the files in the directory /input on HDFS
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples
-2.6.5.jar wordcount /input /output # run the famous hadoop example
hdfs dfs -ls /output # you should see all the outputs
hdfs dfs -copyToLocal /output # download the result from HDFS
```

(j) Stop Hadoop

```
hadoop-daemon.sh stop namenode
hadoop-daemon.sh stop datanode
hadoop-daemon.sh stop secondarynamenode
yarn-daemon.sh stop resourcemanager
yarn-daemon.sh stop nodemanager
mr-jobhistory-daemon.sh stop historyserver
```

**Compile a Hadoop Application**:

(a) Compile (the command "`hadoop classpath`" returns all the dependencies required by Hadoop):
`javac -classpath `hadoop classpath` -d build src/CountingIndexer.java`
The backtick (`) symbol is the same key as the tilde (~) and is located on the top-left of your keyboard.

(b) Jar the Hadoop application: (You can remove the v in -cvf for a clean and quite output)
`jar -cvf CountingIndexer.jar -C build/ .`

**Run a Hadoop Application**:

(a) First, make sure you have uploaded the input files onto HDFS.

(b) Second, use the command to remove the output directory before the next run; otherwise, Hadoop will not override the output directory and hence stop the job.
`hdfs dfs -rm -r /output`

(c) Finally, use the command to run your Hadoop application. The fourth field (which is CountingIndexer in the example) is the class that contains the main method.
`hadoop jar CountingIndexer.jar CountingIndexer /input /output`

**Links:**
*Hadoop:* `http://hadoop.apache.org/`

*Setup Hadoop:*
`http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html`

*Tutorial for Developing a Hadoop Application:*
`http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html`

*MapReduce paper:* `http://research.google.com/archive/mapreduce.html`

*Pride and Prejudice:*
`https://drive.google.com/file/d/0BzxHmFFE_U3Ha2dSOTMwdHJLSzQ/view?usp=sharing`