BIG DATA ANALYTICS
PROF. DR. MARTIN THEOBALD & DR. VINU VENUGOPAL
SUMMER SEMESTER 2021 – UNIVERSITY OF LUXEMBOURG
**Getting Started – February 25, 2021**.

## GET ACCESS TO THE IRIS CLUSTER

**Step 1.**

- Register your student group (up to 3 students) by sending an email to `bigdataanalytics@uni.lu`.
- Refer to "Get an Account" at `https://hpc.uni.lu/users/get_an_account.html` and request access to the IRIS cluster by registering with your student email address and mentioning the "Big Data Analytics" lecture in the "Account Request Form".
- Refer to "Quickstart Guide" at `https://hpc.uni.lu/users/quickstart.html` for an overview of accessing and using the cluster.
- Refer to "Cluster Access" at `https://hpc.uni.lu/users/docs/access.html` for details on how to log in to the cluster using `ssh`; specifically access the IRIS cluster also for the following steps.
- Refer to "File Transfer" at `https://hpc.uni.lu/users/docs/filetransfer.html` for details on how to move files to the cluster using `scp` or `rsync`.
- Refer to "SLURM Launcher Examples" at `https://hpc.uni.lu/users/docs/slurm_launchers.html#apache-spark` for launching Apache Spark on the IRIS cluster.
  - The two scripts `launch-spark-shell.sh` and `launch-spark-submit.sh` provided on Moodle are based on the above launcher script to give you access to a default Spark configuration on IRIS with 2 nodes and 12 CPUs each.
  - Upload the two launcher scripts together with the `SparkWordCount.jar` file (also provided on Moodle, using either `scp` or `rsync`) to your IRIS home directory to run the `SparkWordCount.scala` example via the `launch-spark-submit.sh` script.
  - To run the jar file via `spark-submit` execute:
    ```
    (access-iris)$ sbatch launch-spark-submit.sh
    ```
  - To run a shell script via `spark-shell` execute:
    ```
    (access-iris)$ srun -p batch --time=00:30:0 -N 2 -c 12 --pty bash -i
      (iris-node)$ ./launch-spark-shell.sh
    ```
    (Note that you may as well need to change the file permission using `chmod +x launch-spark-shell.sh` before running the script.)

## SET UP YOUR LOCAL WORKING ENVIRONMENT

**Step 2.**

1. Download and install Java 8 (JDK 1.8) by choosing the respective installer binaries for your system from:
   `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`
   - Linux users can directly download the `jdk-8u<version>-linux-x64.tar.gz` package, unpack the tar archive, and install the JDK with: `tar -zxvf jdk-8u<version>-linux-x64.tar.gz`
   - Windows users have to run the installation binaries and then add the `bin` folder of their JDK installation to their `PATH` variable under the *Windows System Variables.*
   - MacOS users can simply click on the `.dmg` file and install the application.

   Make sure that your `JAVA_HOME` environment variable points to your installation directory of Java.

2. Download and install Scala 2.11.8 by choosing the respective installer binaries for your system from: `https://scala-lang.org/download/2.11.8.html`

   Make sure that your `SCALA_HOME` environment variable points to your installation directory of Scala. Additionally verify that your `PATH` environment variable contains the `bin` directory in which your `scala` and `scalac` binaries are located.

3. Download and install Hadoop 2.7.7 by choosing a respective mirror with the `hadoop-2.7.7.tar.gz` binaries from: `https://archive.apache.org/dist/hadoop/common/hadoop-2.7.7/`

   Unpack the tar archive and make sure that your `HADOOP_HOME` environment variable points to your installation directory of Hadoop. Additionally verify that your `PATH` environment variable contains the `bin` directory in which your `hadoop` binary is located.

4. Download and install Spark 2.2.3 by choosing the `spark-2.2.3-bin-hadoop2.7.tgz` binaries from: `https://archive.apache.org/dist/spark/spark-2.2.3/`

   Unpack the tar archive and make sure that your `SPARK_HOME` environment variable points to your installation directory of Spark. Additionally verify that your `PATH` environment variable contains the `bin` directory in which your `spark-shell` and `spark-submit` binaries are located.

5. Download and install a recent Eclipse IDE for Java Developers by following the steps from the installation guide: `http://www.eclipse.org/downloads/eclipse-packages/`

6. Optionally also download and install a recent Eclipse IDE for Scala by following the steps from the installation guide: `http://scala-ide.org`

   Make sure that you set your Scala Compiler to "Latest 2.11 bundle (dynamic)" when creating a new project in your Eclipse IDE for Scala.

Note: Only extract the Hadoop and Spark packages into a local directory on your computer and make sure that the above environment variables are set properly. You do not actually have to configure a Hadoop or Spark cluster on your own computer!

## Run the WordCount Examples in Apache Hadoop

**Step 3.**

- Download the `Wikipedia-En-41784-Articles.tar.gz` file from Moodle and extract the tar.gz archive into a local directory. Consider only one of the subdirectories (e.g., `AA`) or even just a single file (e.g., `AA/wiki_00`) to get started.

- Download the `HadoopWordCount.java`, `HadoopWordPairs.java` and `HadoopWordStripes.java` classes from Moodle and put them into a new Java project in your Eclipse IDE for Java.

- Import the `hadoop-common-2.7.7.jar` and `hadoop-mapreduce-client-core-2.7.7.jar` libraries from your local Hadoop installation as external jars into the Java build path of your Eclipse project.

- Export your Java project into a single jar file called `HadoopWordCount.jar` from your Eclipse IDE to a local folder.

- Run the `HadoopWordCount` example by typing: `hadoop jar /your/path/to/HadoopWordCount.jar HadoopWordCount <input_directory> <output_directory>`

  The `<input_directory>` should be the location of the Wikipedia articles.

  The `<output_directory>` will be created during the execution of the program and *must not yet exist* in your file system.

- Do the same for the `HadoopWordPairs` and `HadoopWordStripes` examples in your jar file.

## Run the WordCount Example in Apache Spark

**Step 4.**

- Consider once more the `Wikipedia-En-41784-Articles.tar.gz` file from Moodle for this setup.
- Download the `SparkWordCount.scala` class from Moodle and put it into a new Scala project in your Eclipse IDE for Scala.
- Import the `spark-core_2.11-2.2.3.jar`, `spark-tags_2.11-2.2.3.jar`, `hadoop-common-2.7.3.jar`, `hadoop-annotations-2.7.3.jar` and `jackson-annotations-2.6.5.jar` libraries from your local Spark installation as external jars into the Scala/Java build path of your Eclipse project.
- Export your Scala project into a new jar file called `SparkWordCount.jar` from your Eclipse IDE to a local folder.
- Run the `SparkWordCount` example by typing: `spark-submit --class SparkWordCount /your/path/to/SparkWordCount.jar <input_directory> <output_directory>`
- Alternatively try to also execute the Spark shell script `SparkWordCount-shell.scala` from Moodle by calling `spark-shell` of your local Spark installation. Modify the input- and output-directories accordingly and then just manually copy-paste the script into the Spark shell.
- Try to repeat the latter steps on the IRIS cluster by copying the `SparkWordCount.jar` you created previously onto the IRIS cluster and by using the launcher scripts described in Step 1.

## Optional: Run a First Scalability Test

**Step 5.**

- Modify either the `HadoopWordPairs.java` or `HadoopWordStripes.java` classes in your Eclipse IDE for Java, such that they take word neighbours with a distance of up to 1, 10 and 20 words per line into account. Run your modified programs and compare the runtime and the size of the output (i.e., the number of word pairs emitted to the files in your output directory) at each step in Hadoop.

  What conclusion can you draw regarding to the *scalability* of your programs with respect to increasing word distances?