# List of Tutorials

A Scala Tutorial for Java programmers - <http://www.scala-lang.org/docu/files/ScalaTutorial.pdf>

Scala Tutorial - <http://www.tutorialspoint.com/scala/>

Installing Raspbian with NOOBS - <https://www.raspberrypi.org/learning/noobs-install/>

Apache Spark/Raspberry Pi - <https://darrenjw2.wordpress.com/2015/04/17/installing-apache-spark-on-a-raspberry-pi-2/>

Scala/Apache Spark SQL - <http://blog.antlypls.com/blog/2016/01/30/processing-json-data-with-sparksql/>

Hadoop/Raspberry Pi - <https://developer.ibm.com/recipes/tutorials/building-a-hadoop-cluster-with-raspberry-pi/>

# Technologies Used

Raspberry Pi - <https://www.raspberrypi.org/>

Apache Spark - <http://spark.apache.org/>

Apache Spark - <http://www.infoq.com/articles/apache-spark-introduction>

Hadoop - <http://hadoop.apache.org/>

# Design Decisions

## Apache Spark over Hadoop

The Raspberry Pi environment is limited in system resources, specifically RAM. It only has 1GB of RAM which is partitioned for use between the system (accessible by the user) and the GPU.

The Raspberry Pi’s were configured with the least amount of RAM for the GPU (16MB), leaving the remaining space available for user applications. They were not going to run any sort of GUI or windowing system (like the X Window System), so any larger allocation of RAM for GPU use was unnecessary.

The Raspbian Linux operating system (OS) was installed on the Raspberry Pi’s, as it has been tailored to best utilise the hardware’s resources and features. Further configuration was done on the OS to disable the daemons responsible for controlling the on-board Bluetooth and Wi-Fi modules, thus freeing up even more available RAM.

Linux’s Network File System (NFS) service was installed and enabled on the Raspberry Pi master node so the slaves in the Spark configuration could operate on a single source of data, with the least amount of memory overhead. When using HDFS, the RAM used by the slave nodes increased from 99MB to 125MB, and from 181MB to 206MB on the master node. In addition to extra RAM use, the time to upload the source data (1GB of plain text files) to HDFS took around 8 minutes, as the data was partitioned across all the data nodes within the cluster.

Table 1: RAM use per system

|  |  |  |
| --- | --- | --- |
| System | RAM used (in MB) | RAM used (%) |
| Raspbian OS | 35 | 3.6 |
| Spark (master) | 170 | 17.47 |
| Spark (slave) | 115 | 11.82 |
| Hadoop (master) – Empty HDFS | 181 | 18.6 |
| Hadoop (slave) – Empty HDFS | 99 | 10.17 |
| Hadoop (master) – Populated HDFS | 206 | 21.17 |
| Hadoop (slave) – Populated HDFS | 125 | 12.85 |

When comparing the amount of RAM available for map-reduce applications and the memory required to work with datasets, Apache Spark was the primary candidate due to its RAM utilisation.

Table 2: RAM footprints per system

|  |  |
| --- | --- |
| System | RAM footprint (%) |
| Spark (master) | 21.06 |
| Hadoop (master) – Populated HDFS | 24.77 |
| Spark (slave) | 15.42 |
| Hadoop (slave) – Populated HDFS | 16.44 |