Big Data Case Study: Spark on Armv8

Overview

During the last years different *big data* technologies have been developed and are playing a critical role in science and business. In this paper, we examine the readiness of Arm®-based platforms for two of the more prominent *big data* technologies namely Spark and Hadoop. For the benchmark we installed Spark over Hadoop HDFS and executed some generic mathematical examples and a specific genomic pipeline developed in the HPG BigData project from OpenCB platform (https://github.com/opencb). OpenCB platform consists of several repositories for big data analysis and visualization in genomics. For server hardware, we leverage the ThunderX1® platform from Cavium®, with a highly integrated SoC architecture and a range of configurations addressing modern data centre and cloud implementations[1]. One of our goals is to test the scaling of the Cavium Thunder X1, as a platform for future production.

Architecture

Cavium ThunderX is a System on Chip (SoC) based on the Armv8 architecture. A single socket is comprised of 48 physical, fully out-of-order cores with up to four DDR4 memory controllers. A compute node typically has two sockets for a total of 96 cores and up to 1TB of memory. Our cluster consists of one login node and four slaves nodes. Each slave node used in this study was configured with 128GB of DDR4 memory and a 40Gb Ethernet interconnect.

Both Spark and HDFS have a master and slave architecture. Spark slaves are called workers; the HDFS master is called the *namenode* and HDFS slaves are called *datanodes*. In our cluster, the Spark master and the HDFS *namenode* run on the login node and Spark workers and HDFS *datanodes* run on the slaves nodes.

Software ecosystem

Our study ran under the Ubuntu 16.04 operating system, and with the following software: Oracle Java JDK 1.8.131, Scala, OpenSSH (both server and client), Hadoop HDFS (vanilla) 2.7.3 and Spark 2.1. All these packages were very easy to install following the documentation available on internet, and thus the installation worked out-of-the-box.

Spark[2] is a fast and general engine to process large-scale datasets and can run on Hadoop, standalone, or in the cloud, and it can access diverse data sources including HDFS, HBase, and S3. In addition, Spark provides four high-level distributed libraries: **Spark SQL**, a distributed SQL query engine; **MLlib**, a distributed machine learning library that includes classification and regression methods, clustering algorithms or principal component analysis; **Spark Streaming**, a distributed

[1] cavium.com/ThunderX_Arm_Processors.html
[2] https://spark.apache.org/

library that enables scalable, high-throughput, fault-tolerant stream processing of live data streams; **GraphX**, a distributed graph processing framework.

HDFS[3] is the Hadoop Distributed File System. HDFS provides high-throughput access to huge data by storing data across multiple machines enabling fast and parallel access to data making applications available for parallel processing and is also highly fault-tolerant.

Avro[4] is widely used as a common serialization platform, as it interoperable across multiple languages, offers a compact and fast binary format. Avro offers complex data structures representation defined by schemas. In addition, Avro is splittable that makes it highly suitable for the map-reduce paradigm. **Parquet**[5] is a columnar storage format compatible with Hadoop/Spark data processing frameworks. Parquet is more efficient than Avro in terms of data compression and encoding schemas.

The use of these Big Data technologies provides a highly efficient and intuitive method for supporting Genomic analyses.

OpenCB provides advanced open-source software for the analysis of high-throughput genomic data. During recent years advances in high-throughput sequencing technologies have produced data at an unprecedented scale and bioinformaticians encounter difficulties in storage and analysis of vast amounts of biological data. Biology has joined the *big data* era and faces new challenges such as the storage, analysis, search, sharing and visualization of data that require new solutions. OpenCB supplies a solution to this.

Spark application performance

Generic examples

Before running the genomic specific application we studied the performance of the built-in examples of Spark with 2 and 4 slaves. These examples are *SparkPi* that computes an approximation to Pi; *SparkALS* for alternating least squares matrix factorization; and *SparkTC* that computes the transitive closure on a graph. All these examples are available at Spark's GitHub repository[6].

[3] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[4] https://avro.apache.org/
[5] https://parquet.apache.org/
[6] github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples

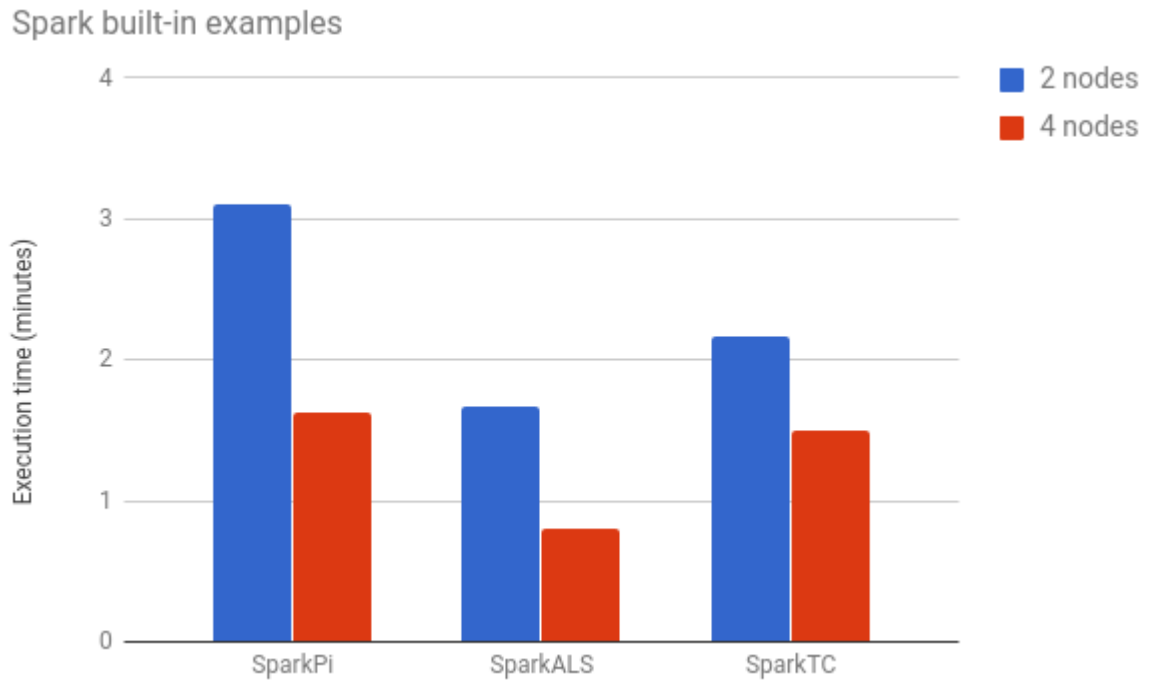
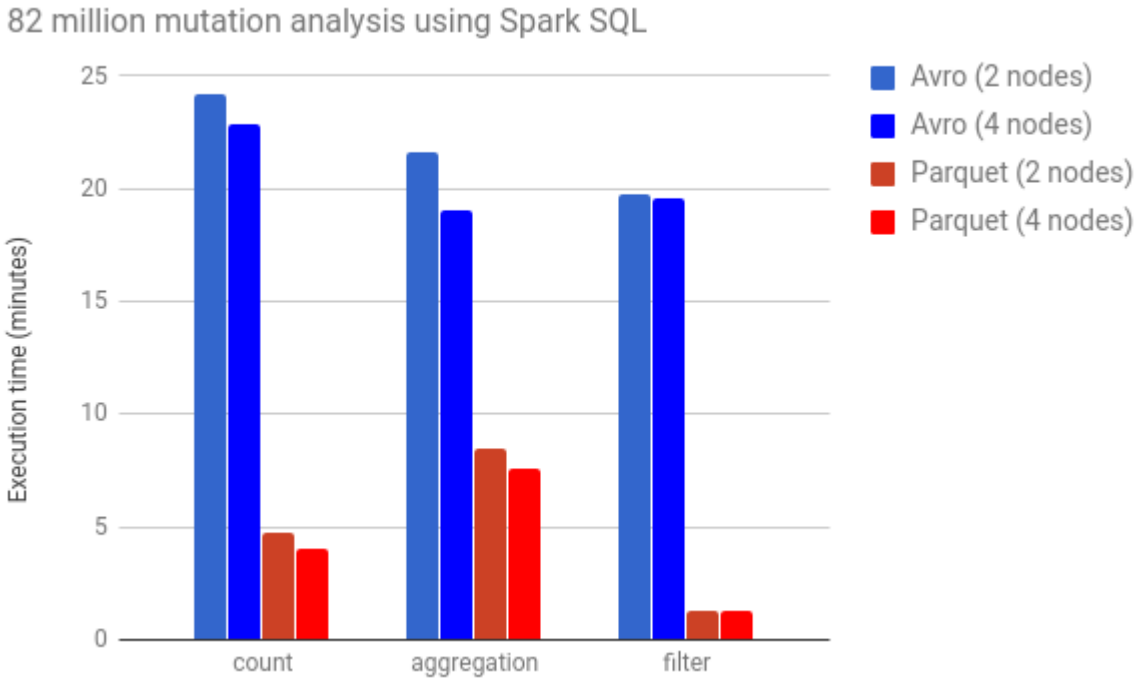


Figure 1. Three Spark built-in examples with two and four servers

Studying the scalability was not a main goal of this paper given the small size of the cluster. However we decided to execute these examples with 2 and 4 servers. These examples are mainly CPU-bound problems, and as can be observed in Figure 1 the performance with the four servers improves significantly compared to the execution with two servers, being almost two times faster. Although the results look promising a bigger cluster is needed to study performance and scalability in more detail.

Genomic use case

Many diseases are caused by mutations affecting genes or genomic regulatory regions, these mutations can potentially affect when the gene is active or can even modify the protein sequence, in both cases the function of the gene or protein is altered and this malfunction can cause a disease, currently there are thousands of genetic diseases known such as diabetes or cancer. Current genomic data take few GB per sample, this data can be processed and analyzed with big data technologies such as Spark. Three common genomic analyses were executed over a dataset with 82 million genomic mutations using Spark SQL. This dataset was stored in both Avro and Parquet file formats compressed with *deflate* algorithm to study the performance with both formats. The executed analysis included counting the number of genomic variants, executing a *group by* aggregation and genomic query filter.



Comments:

1. As expected Parquet executions are faster than Avro. Parquet columnar storage format makes Spark SQL queries more efficient especially for queries that only address a reduced number of columns (e.g. the genetic filter analysis). In general, Parquet reduces storage size and transfer costs.
2. Surprisingly no performance improvement was observed with four nodes when compared to two nodes. One possible explanation is that Avro and Parquet files are not well distributed across the HDFS servers and therefore they are not being processed by all Spark nodes, performance improvements should be seen with larger files and a larger Spark cluster.
3. Performance with both Avro or Parquet is, however, still superior when compared to current tools in genomics that work with VCF file in a single node.

Conclusion

Arm Cavium has shown enough compute capacity to form a cluster and analyse a non-trivial size dataset. Installing and executing Hadoop HDFS and Spark on the Arm Cavium X1 was straightforward, pleasing as they are two of the better known *big data* technologies.

During the benchmarks we observed up to a 2x performance improvement for CPU-bound problems when executing from two to four nodes, as the job scaled. However, when executing the genomic analysis over Avro and Parquet files we could not see any performance improvement between two and four nodes: this is probably related with a sub-optimal file chunk distribution across the HDFS servers. With larger files and a larger cluster we anticipate that scalability will be maintained.

In summary, Spark over the Cavium ThunderX1 looks very promising, all implemented jobs run smoothly without any issue, and most showed very good performance. We think Arm Cavium offers a very attractive solution for Big Data analysis with Spark.