Good parallel software development practices. Apache Spark case

Myroslava Stavnycha
Faculty of Software Engineering
University of Tartu

***Abstract*—Recently, Spark as data processing engine, gained huge popularity because of better performance in terms of the speed. Developers of Spark claim that it may outperform Hadoop MapReduce in 100 times in memory and 10 times on disk [1]. This paper outlines which innovations improved speed and how. In order to investigate improvements, I analysed technical documentation, which is available, since both projects are open-source, and tested performance by myself. Indeed, not only the ability of Spark to run in-memory influences the speed, but also Spark enhanced architecture, so set of MapReduce overheads were eliminated.**

I. Introduction

Hadoop Spark appeared in 2009 and instantly became the best project among its alternatives. It improves MapReduce design pattern and allows to perform tasks on 1TB of data during 5-7 seconds. MapReduce is one of key programming models among structural design patterns. It is composed of 2 stages:

1) Map
2) Reduce.

On the first stage one single function is mapped onto independent sets of data. On the second stage the results of mapping are reduced. It evolved from Hadoop MapReduce, but, in contrast to MapReduce, it includes many components besides Core Engine:

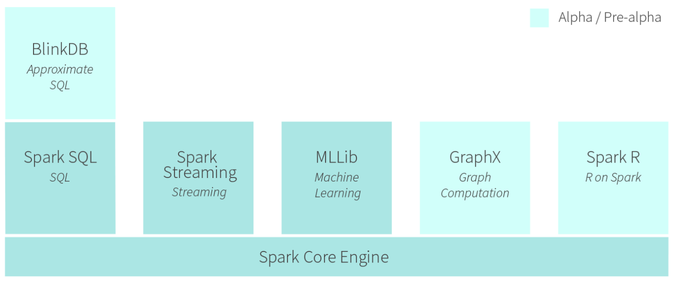

Fig. 1. Spark components

Spark Core is underlying execution engine, which supports wide variety of applications and APIs for Scala, Python and Java for ease of development.

Spark MLLib is scalable and fast machine learning library, that provides high-quality algorithms.

Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's simplicity of use and fault tolerance features.

II. Hadoop Spark as improvement

Codebase of Hadoop Spark is 20,000 LOC in comparison with Hadoop 2.0, which has 220,000 LOC. Spark is not a refactored MapReduce, but completely a new framework, which is built atop of MapReduce. The solution not to touch MapReduce was derived from the fact that redesigning of a whole framework without breaking applications, which depend on it, is a huge risk. Choosing this way would mean several years of improving the system before it can be completely reliable in production. However, creating completely new product without old technical debts and old dependancies, designed for next generation workloads, is much more promising solution, easier and less risky [2].

Spark is written in Scala programming language, because Scala can easily manipulate with distributed datasets as with locally collective objects. Moreover, Scala is extensible, multi-paradigm language, which runs directly on Java Virtual Machine. This allows Scala to run everywhere, where JVM runs.

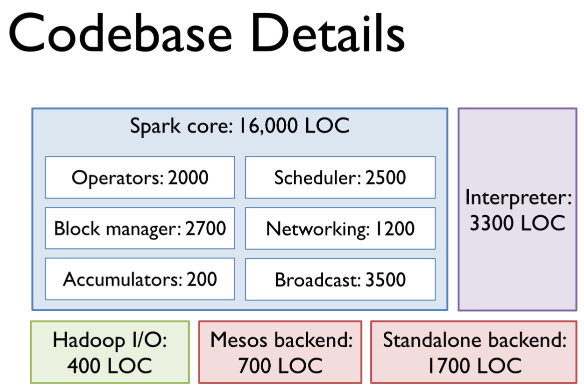

Fig. 2. Spark codebase

III. Spark community

Spark was developed in 2009 and now it has 32 commiters as representatives from different organisations (Databricks, Intel, Yahoo!, UC Berkley) and many more contributors. In order to become a commiter, person must have a history of big contribution to Spark. Currently, team regularly votes for adding new commiters. Spark has a huge list of massive users, like UC Berkley AMPLab (data research lab, that initially launched Spark), Amazon, Baidu, eBay Inc (logging of transaction aggregation and analytics), NASA JPL, Samsung Research America, Shazam Entertainment, Yahoo!, Yandex and many others [3].

The system was developed in Berkeley, CA and currently their office is situated there. Databricks is commercialising Spark project, involving such investors as: Andreessen Horowitz, NEA. The creator of Spark is Matei Zaharia, assistant professor of Computer Science at MIT.

IV. Hadoop MapReduce community

Hadoop MapReduce was developed in 2005 by Doug Cutting and Mike Cafarella, and it is a core project of Apache. Since MapReduce is one of the core modules of Hadoop, all Apache Hadoop community is involved into improving this project. Project Management Committee of Hadoop contains 50 members from different organisations, including Cloudera, Yahoo!, Facebook, LinkedIn, Twitter and VMware. In addition, project involves 70 commiters all over the world. MapReduce is funded through The Apache Software Foundational Sponsorship Program, where everybody can donate. Moreover it has tenths of platinum sponsors, including Yahoo!, Google, Citrix, Microsoft, Facebook and HP. The system has its own list of massive users, including LinkedIn, Last.fm, Facebook, Spotify, Twitter and others [4].

V. Spark vs MapReduce

MapReduce is highly scalable, but it suffers from a critical weakness for machine learning: it does not support iteration.

Hadoop weakness is the inability to reuse intermediate results across several simultaneous computations. That is an issue, which researchers at the University of California Berkeley's AMP lab were trying to conquer with the open source project Spark.

It is possible to develop distributed machine learning algorithms on the classic MapReduce computation framework in Hadoop (e.g. Apache Mahout). But MapReduce is notoriously low-level and difficult to express complex computations in.

Hence, the main advantage of Hadoop Spark is that it is applicable for iterative algorithms, which are the huge segment of machine learning area. When similar function is applied to the same dataset, the same data is shared and reused all the time. Being able to use that efficiently is crucial for performance. MapReduce reads the data from stable storage system on each step and puts it back after processing for the next step. In fact, it is the only way to pass the data between iterations with MapReduce approach. However, more than 90% can be spent on performing all serialization and deserialization steps or other actions, associated with storage system. One can perfectly scale the application on set of clusters, but the system will still work 10 times slower than it could do. For solving this problem several systems were introduced, like Pregel and HaLoop, which keep intermediate data during computations in memory and offer iterative MapReduce interface. But those systems support only specific computational patterns and perform data sharing implicitly for those patterns. They do not provide abstraction for more general reuse [5].

Moreover, original MapReduce executes jobs in a simple, transparent but rigid structure: a processing or transform step ("map"), a synchronisation step ("shuffle"), and a step to combine results from all the nodes in a cluster ("reduce"). For more sophisticated computations one should collect together a join of MapReduce jobs and execute them in sequence. Each of those jobs is high-latency, and none usually can start until the previous job has finished completely. Consequently, complex, multi-stage applications are relatively slow.

An alternative approach was introduces to let developers construct complex, multi-step directed acyclic graphs (DAGs) of work, and to execute DAGs all at once, simultaneously. This eliminates the costly synchronisation required by MapReduce and makes applications easier to build. Prior research on DAG engines includes Dryad, a Microsoft Research project used internally at Microsoft for its Bing search engine and other hosted services.

Spark builds on those ideas but adds some important innovative features. For instance, Spark supports in-memory data sharing across DAGs, so that different jobs can work with the same data at higher speed. It also allows cyclic data flows. As a result, Spark handles iterative graph algorithms (social network analysis), machine learning and stream processing extremely well. Those have been cumbersome to build on MapReduce and even on other DAG engines [2].

In addition, Matei Zaharia claims, that a key design choice for building Spark was using Scala, which he describes as a "high level language that allows concise, functional programming" [5]. The project took off when Zaharia and his collaborators found that Scala's interactive shell could be modified to run iterative data mining algorithms.

Eventually, Spark has following major changes:

1) It can store data in-memory;
2) Resilient Distributed Dataset as new main abstraction was introduced. Moreover, RDD provides user friendly API for data processing, which supports all necessary operations.
3) Apache Spark is supported by Scala, Java and Python.

VI. Storing data

Hadoop does everything on disk via HDFS, whereas Spark has number of persistence configuration, which can be set separately with respect to every data object. The default value of persistence is MEMORY_ONLY, which means that all data RAM is kept always in memory. In case of insufficient memory lacking data components are recomputed again. This functionality is supported by Spark conception of Resilient Distributed Dataset, which is described below. In this case Spark eliminates a lot of Hadoop's overheads, such as the reliance on I/O for everything.

Another persistence configuration is MEMORY_AND_DISK, which means that in case of insufficient memory data is spilled on the file system.

Hence, the primary reason to use Spark is a speed, and this comes from the fact that its execution can keep data in memory between stages rather than always persist back to HDFS after a Map or Reduce. This advantage is very pronounced for iterative computations, which have tens of stages, each of which is touching the same data. This is where things may be 100 times faster. Apparently, one-pass ETL-like job for which MapReduce was designed, is not in general faster.

VII. Resilient Distributes Dataset

Spark is growing around the concept of a Resilient Distributed Dataset (RDD), which is a collection of elements that are processed in parallel. RDD supports 2 types of operations:

1) Transformation: operation that returns value after a computation on dataset (map).
2) Aggregation: aggregates all the elements of RDD and returns the result to the driver program (reduce).

RDD provides rich and flexible API for work with distributed dataset. API contains all necessary operations, such as: map, filter, union, intersection, distinct, join, reduce, collect and others. In contrast, Hadoop has strict and rigid API, which doesn't allow such versatility. In addition, scripts for Hadoop require usually more lines of code, because it doesn't have so rich API.

Moreover, RDD is fault-tolerant dataset. In case of loosing some part of data, RDD knows how to recompute it: from which data it was derived and which function to apply again, in order to restore the loss.

In case of insufficient RAM some partitions of RDD will not be cached, but will be recomputed each time they are requested. This is faster solution unless one has heavy computational function.

If developer wants to persist data on the file system, he can configure the application with parameters, which Spark provides for persisting:

1) MEMORY_ONLY: Default value. Stores data as deserialised objects in JVM memory and in case of insufficient memory, recomputes on a fly some dataset, that can't be cached.

2) MEMORY_AND_DISK: Stores data as deserialised objects in JVM memory and in case of insufficient memory, spills data to the file system.

3) MEMORY_ONLY_SER: Stores data as serialised objects in memory. More sufficient in terms of space and preserves the speed of access to data. In case of insufficient memory, recomputes datasets every time they are requested.

4) MEMORY_AND_DISK_SER: Similar to MEMORY_ONLY_SER, but stores data on disk in case of lack of memory.

5) DISK_ONLY: Always stores data on disk.

Moreover, in case of time-consuming long lineage recomputing of RDD component, Spark provides a Checkpoint technology, which allows to save current state of RDD component to a stable storage. Thus, in case of loss of data slice, Spark will not spend time on recomputing it, but will restore from the data set. The checkpoint creation is a duty of a user, instead Spark provides API for it.

VIII. Comparison

Picture below describes the difference in performance of Spark and Hadoop MapReduce, running on a system with 1 node:

1) Intel Core i5, 2.40GHz;
2) 8GB of RAM;
3) Hadoop version 2.5.1;
4) Spark version 1.1.0.

The repository of Apache Spark can be found at [6]. The repository of Apache Hadoop is placed at [7].

Standard implementation of counting words was used as algorithm, that was running on the data of different sizes, using RAM of 8 GB.

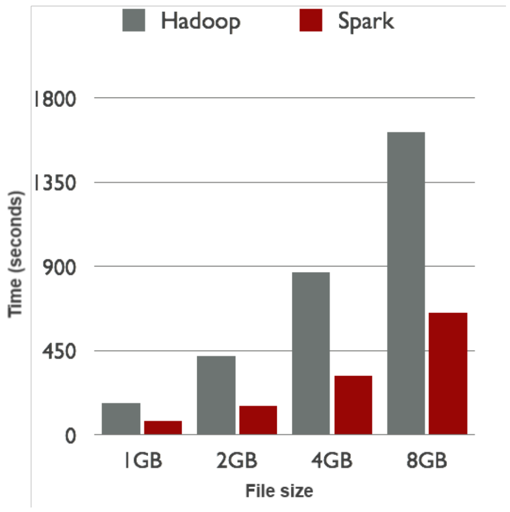

Fig. 3. Counting words in documents of different sizes. 1 node, MEMORY_ONLY

Basically, this is not the best algorithm to demonstrate 100 times better performance of Spark [1], but it still outperformes MapReduce in 2-3 times.

On Figure 4 different configurations of data persistence are described. Despite the fact, that Spark DISK_ONLY and Hadoop MapReduce manipulate with data persistence in the same manner, Spark still shows better results.

Similarly, the same test was performed on the cluster with 4 nodes:

1) 2GB of RAM per node;
2) 1 CPU per node;
3) Hadoop version 1.2.1;
4) Spark version 1.1.0.

More information about cluster can be found from [8]. Hadoop and Spark were installed on SLURM using framework available at [9].

During this test Spark shows better performance even with DISK_ONLY persistence configuration, as it is described on Figure 5.

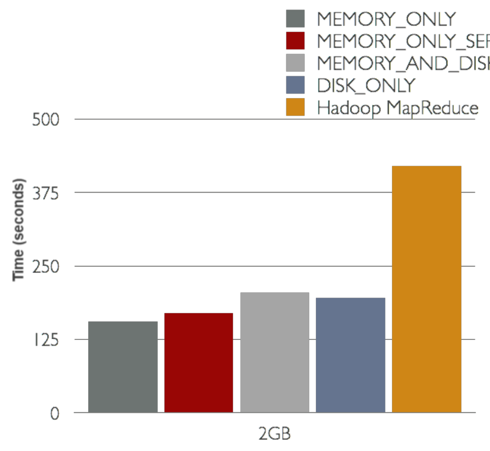

Fig. 4. Spark, different configurations of data persistence and Hadoop MapReduce

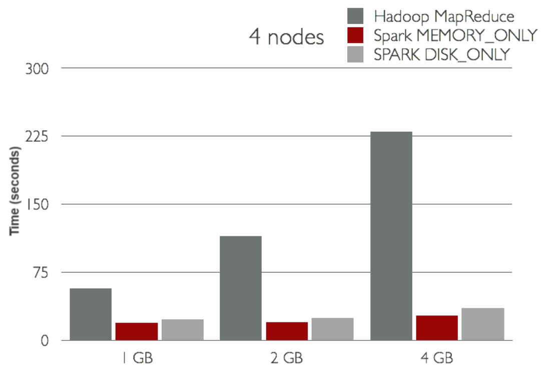

Fig. 5. Spark and Hadoop MapReduce performance on cluster with 4 nodes

The explanation lies is several factors:

1) Overhead of Hadoop software stack;
2) Signalling overheads of datanodes to namenode;
3) Overheads of HDFS manipulating data;
4) Deserializaiton overheads of binary data to java objects [5].

Factors, mentioned above, describe many aspects of HDFS work. HDFS service data includes:

1) Monitoring;
2) Rebalancing;
3) Managing integrity;
4) Metadata replication;
5) Fault tolerance [10].

Monitoring is continuous ”heartbeating” of datanodes to master. If master node doesn’t receive a signal from worker, it is assumed to be unavailable.

Rebalancing is migrating of a data component from one node to another for improving further performance.

Managing integrity is support of digital signature, checksum, for data component.

Metadata replication is replication of metadata, which also can be lost, in order to support fault tolerance.

IX. Summary

This paper outlines main advantages of Spark over Hadoop MapReduce and lists main improvements that Spark embodied. Tests, performed on cluster with 4 nodes, showed precedence of Spark regarding the speed. However, with chosen algorithm performance sped up up to 10 times using in-memory computation.

References

[1] (2009) Official spark documentation. [Online]. Available: https://spark.apache.org/documentation.html
[2] M. Olson, “Mapreduce and spark,” *Cloudera blog*, 2013.
[3] Companies and organizations. [Online]. Available: https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark
[4] Who are we. [Online]. Available: http://hadoop.apache.org/who.html
[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.
[6] Apache spark github repository. [Online]. Available: https://github.com/apache/spark
[7] Apache hadoop github repository. [Online]. Available: https://github.com/apache/hadoop
[8] Rocket cluster. [Online]. Available: http://hpc.ut.ee/rocket_cluster
[9] G. Lockwood. Framework for launching hadoop clusters on slurm. [Online]. Available: https://github.com/glennklockwood/myhadoop
[10] D. Loshin. (2013) Understanding the big data stack: Hadoops distributed file system. [Online]. Available: http://data-informed.com/understanding-the-big-data-stack-hadoops-distributed-file-system/