Survey on Frameworks for Distributed Computing: Hadoop, Spark and Storm

Telmo da Silva Morais

Student of Doctoral Program of Informatics Engineering
Faculty of Engineering, University of Porto
Porto, Portugal
Telmo.morais@gmail.com

Abstract
The storage and management of information has always been a challenge for software engineering, new programing approaches had to be found, parallel processing and then distributed computing programing models were developed, and new programing frameworks were developed to assist software developers. This is where Hadoop framework, an open source implementation of MapReduce programing model, that also takes advantage of a distributed file system, takes its lead, but in the meantime, since its presentation, there were evolutions to the MapReduce and new programing models that were introduced by Spark and Storm frameworks, that show promising results.

Keywords: Programing framework, Hadoop, Spark, Storm, distributed computing.

1 Introduction

Through time, size of information kept rising and that immense growth generated the need to change the way this information is processed and managed, as individual processors clock speed evolution slowed, systems evolved to a multi-processor oriented architecture. However there are scenarios, where the data size is too big to be analysed in acceptable time by a single system, and in this cases is where the MapReduce and a distributed file system are able to shine.

Apache Hadoop is a distributed processing infrastructure. It can be used on a single machine, but to take advantage and achieve its full potential, we must scale it to hundreds or thousands of computers, each with several processor cores. It's also designed to efficiently distribute large amounts of work and data across multiple systems.

Apache Spark is a data parallel general-purpose batch-processing engine. Workflows are defined in a similar and reminiscent style of MapReduce, however, is much more capable than traditional Hadoop MapReduce. Apache Spark has its Streaming API project that allows for continuous processing via short interval batches. Similar to Storm, Spark Streaming jobs run until shutdown by the user or encounter an unrecoverable failure.

Apache Storm is a task parallel continuous computational engine. It defines its workflows in Directed Acyclic Graphs (DAG's) called "topologies". These topologies run until shutdown by the user or encountering an unrecoverable failure.

1.1 The big data challenge

Performing computation on big data is quite a big challenge. To work with volumes of data that easily surpass several terabytes in size, requires distributing parts of data to several systems to handle in parallel. By doing it, the probability of failure rises. In a single-system, failure is not something that usually program designers explicitly worry about.[1]

However, in a distributed scenario, partial failures are expected and common, but if the rest of the distributed system is fine, it should be able to recover from the component failure or transient error condition and continue to make progress. Providing such resilience is a major software engineering challenge. [1]
In addition, to these sorts of bugs and challenges, there is also the fact that the compute hardware has finite resources available. The major hardware restrictions include:

- Processor time
- Memory
- Hard drive space
- Network bandwidth

Individual systems usually have few gigabytes of memory. If the input dataset is several terabytes, then this would require a thousand or more machines to hold it in RAM and even then, no single machine would be able to process or address all of the data.

Hard drives are a lot bigger than RAM, and a single machine can currently hold multiple terabytes of information on its hard drives. But generated data of a large-scale computation can easily require more space than what original data had occupied. During this, some of the storage devices employed by the system may get full, and the distributed system will have to send the data to other node, to store the overflow.
Finally, bandwidth is a limited resource. While a pack of nodes directly connected by a gigabit Ethernet generally experience high throughput between them, if all transmit multi-gigabyte, they would saturate the switch's bandwidth. Plus, if the systems were spread across multiple racks, the bandwidth for the data transfer would be more diminished [1].

To achieve a successful large-scale distributed system, the mentioned resources must be efficiently managed. Furthermore, it must allocate some of these resources toward maintaining the system as a whole, while devoting as much time as possible to the actual core computation[1].

Synchronization between multiple systems remains the biggest challenge in distributed system design. If nodes in a distributed system can explicitly communicate with one another, then application designers must be cognizant of risks associated with such communication patterns. Finally, the ability to continue computation in the face of failures becomes more challenging[1].

Big companies like Google, Yahoo, Microsoft have huge clusters of machines and huge datasets to analyse, a framework like Hadoop helps the developers use the cluster without expertise in distributed computing, and taking advantage of Hadoop Distributed File System.[2]

2 State of the Art

This section will begin to explain what is Apache's Hadoop Framework and how it works, also a short presentation of other Apache alternative frameworks, namely Spark and Storm.

2.1 The Hadoop Approach

Hadoop is designed to efficiently process large volumes of information by connecting many commodity computers together to work in parallel. One hypothetic 1000-CPU machine would cost a very large amount of money, far more than 1000 single-CPU or 250 quad-core machines. Hadoop will tie these smaller and more reasonably priced machines together into a single cost-effective compute cluster.[1]
Apache Hadoop has two pillars:

- YARN - Yet Another Resource Negotiator (YARN) assigns CPU, memory, and storage to applications running on a Hadoop cluster. The first generation of Hadoop could only run MapReduce applications. YARN enables other application frameworks (like Spark) to run on Hadoop as well, which opens up a wide set of possibilities.[3]
- HDFS - Hadoop Distributed File System (HDFS) is a file system that spans all the nodes in a Hadoop cluster for data storage. It links together the file systems on many local nodes to make them into one big file system.[3]

MapReduce
Hadoop is modelled after Google MapReduce. To store and process huge amounts of data, we typically need several machines in some cluster configuration.

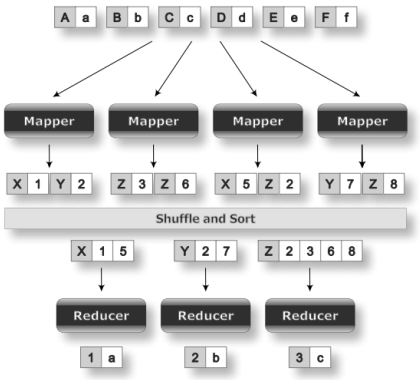

Fig. 1. - MapReduce flow

A distributed file system (HDFS for Hadoop) uses space across a cluster to store data, so that it appears to be in a contiguous volume and provides redundancy to prevent data loss. The distributed file system also allows data collectors to dump data into HDFS, so that it is already prime for use with MapReduce. Then the Software Engineer writes a Hadoop MapReduce job [4].

Hadoop job consists of two main steps, a map step and a reduce step. There may be, optionally, other steps before the map phase or between the map and reduce phases. The map step reads in a bunch of data, does something to it, and emits a series of key-value pairs. One can think of the map phase as a partitioner. In text mining, the map phase is where most parsing and cleaning is performed. The output of the mappers is sorted and then fed into a series of reducers. The reduce step takes the key value pairs and computes some aggregate (reduced) set of data, i.e. sum, average, etc [4].

The trivial word count exercise starts with a map phase, where text is parsed and a key-value pair is emitted: a word, followed by the number "1" indicating that the key-value pair represents 1 instance of the word. The user might also emit something to coerce Hadoop into passing data into different reducers. The words and 1s are sorted and passed to the reducers. The reducers take like key-value pairs and compute the number of times the word appears in the original input.[5]

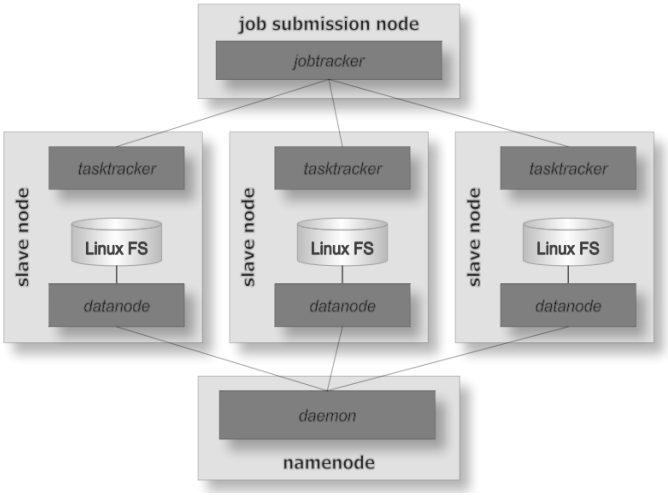

Fig. 2. - Hadoop workflow[2]

2.2 SPARK framework

Apache Spark is an in-memory distributed data analysis platform, primarily targeted at speeding up batch analysis jobs, iterative machine learning jobs, interactive query and graph processing. One of Spark's primary distinctions is its use of RDDs or Resilient Distributed Datasets. RDDs are great for pipelining parallel operators for computation and are, by definition, immutable, which allows Spark a unique form of fault tolerance based on lineage information. If you are interested in, for example, executing a Hadoop MapReduce job much faster, Spark is a great option (although memory requirements must be considered) [19].

It provides high-level APIs in Java, Scala and Python, and an optimized engine that supports general execution graphs.

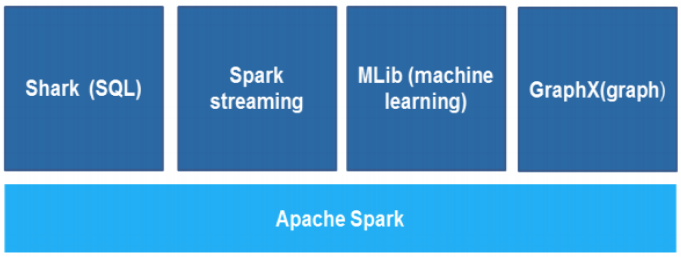

Fig. 3. - Spark Framework

It also supports a rich set of higher-level tools, including Shark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming [6], helping the development of parallel applications.

The main goal of Spark is to work with distributed collections, as you would with local ones. It relays on a resilient distributed datasets (RDDs), that is a immutable collections of objects spread across a cluster, built through parallel transformations (map, filter, etc), automatically rebuilt on failure controllable persistence (e.g. caching in RAM) for reuse, shared variables that can be used in parallel operations [6].

Resilient Distributed Datasets (RDDs)

Spark's main abstraction is resilient distributed datasets (RDDs), which are immutable, partitioned collections that can be created through various data-parallel operators.

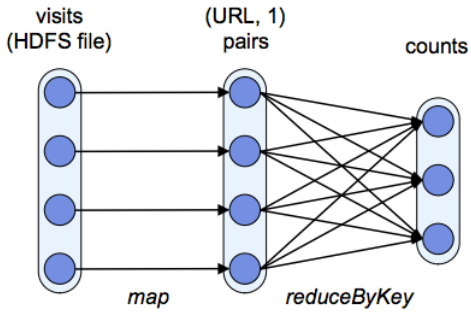

Fig. 4. - Lineage graph for the RDDs in our Spark example.[7]

Each RDD is either a collection stored in an external storage system, such as a file in HDFS, or a derived dataset created by applying operators to other RDDs. For example, given an RDD of (visitID, URL) pairs for visits to a website, we might compute an RDD of (URL, count) pairs by applying a map operator to turn each event into a (URL, 1) pair, and afterward a reduce to add the counts by URL.[7]
Spark provides three options for persist RDDs:

1. In-memory storage as deserialized Java Objects (fastest, JVM can access RDD natively) [2].
2. In-memory storage as serialized data (space limited) [2].
3. On-disk storage (RDD too large to keep in memory, and costly to recomputed) [2].

Spark streaming

The key idea behind the model is to treat streaming computations, as a series of deterministic batch computations, on small time intervals. The input data received during each interval is stored, reliably across the cluster, to form an input dataset for that interval. Once the time interval completes, this dataset is processed via deterministic parallel operations, such as *map*, *reduce* and *groupBy*, to produce new datasets repre-

senting program outputs or intermediate state. It stores these results in resilient distributed datasets (RDDs)[8].

Apache Spark does not itself require Hadoop to operate. However, its data parallel paradigm requires a shared file system for optimal use of stable data. The stable source can be S3, NFS, or, more typically, HDFS) [9].

2.3 Storm Framework

Apache Storm is, a free and open source distributed real-time computation system, focused on stream processing or what some call complex event processing. Storm implements a fault tolerant method for performing a computation or pipelining multiple computations on an event, as it flows into a system. One might use Storm to transform unstructured data, as it flows into a system into a desired format)[9].

Apache Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. Storm has many use cases: real-time analytics, online machine learning, continuous computation, distributed RPC, ETL and more. It's scalable, fault-tolerant, guarantees your data will be processed, is easy to set up and operate [9].

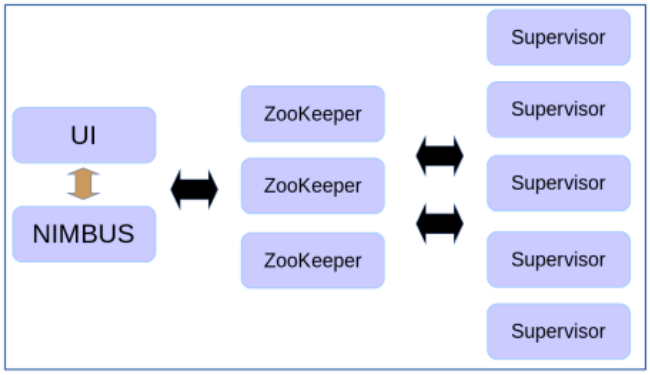

Fig. 5. - Storm Framework system architecture

System architecture:

- Nimbus: Like JobTracker in Hadoop
- Supervisor: Manage workers
- Zookeeper: Store meta data
- UI: Web-UI

A Storm cluster is superficially similar to a Hadoop cluster. Whereas on Hadoop you run "MapReduce jobs", on Storm you run "topologies". "Jobs" and "topologies" themselves are very different; one key difference is that a MapReduce job eventually finishes, while a topology processes messages forever (or until you kill it).

There are two kinds of nodes on a Storm cluster: the master node and the worker nodes. The master node runs a daemon called "Nimbus" that is similar to Hadoop "JobTracker". Nimbus is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures [9].

Each worker node runs a daemon called the "Supervisor". The supervisor listens for work assigned to its machine, starts and stops worker processes, as necessary based on what Nimbus has assigned to it. Each worker process executes a subset of a topology. A running topology consists of many worker processes, spread across many machines [9].

Storm does not natively run on top of typical Hadoop clusters, it uses Apache ZooKeeper and its own master/minion worker processes to coordinate topologies, master and worker state, and the message guarantee semantics [9].

Having said that, both Yahoo! and Hortonworks are working on providing libraries for running Storm topologies on top of Hadoop 2.x YARN clusters.

Regardless, Storm can certainly still consume files from HDFS and/or write files to HDFS[18][21].

3 Discussion

Spark is one of the newest players in the MapReduce field. Its purpose is to make data analytics fast to write, and fast to run. Unlike many MapReduce systems (Hadoop inclusive), Spark allows in-memory querying of data (even distributed across machines) rather than using disk I/O. It's no surprise that Spark out-performs Hadoop on many iterative algorithms. Spark is implemented in Scala, a functional object-oriented language that runs on top of the JVM. Similar to other languages like Python and Ruby, Scala has an interactive prompt that users can use to query big data straight from the Scala interpreter, making it a good choice in some scenarios. However, it does not support a distributed file system on its own, it depends on Hadoop, if a HDFS is required.

The Storm framework is referred as being the Hadoop of Real-time Processing. Hadoop is a batch-processing system, this means, give it a big set of static data and it will do something with it. Storm is real-time, it processes data in parallel as it streams. Therefore, Storm is more a complement to Hadoop rather than a real replacement, as Storm fails when it comes to process large persistent data, as its focus is to be able to process a large number of streams of data (in real time computation), while Hadoop focus is on large amount of persistent data (batch processing).

3.1 Frameworks features summary

In the beginning of this survey, I did not know what I would find on programing frameworks for distributed computing. Therefore, after this review, summarizing the main features and benefits of each of the evaluated frameworks, may serve as contribution to an appropriated and better-informed selection of the framework, that aims

the deployment of a new distributed computing platform, or for those considering to improve an already existing one.

	Storm	Spark Streaming
Processing Model	Record at a time	Mini batches
Latency	Sub second	Few seconds
Fault tolerant – every record processed	At least one (may be duplicates)	Exactly one
Batch framework integration	Not available	Spark
Supported languages	Storm was designed from the ground up to be usable with any programming language [9].	Python, Scala, Java

Table 1. Storm vs. Spark Streaming

Based on my research, the comparison must be made based on use cases oriented view, as the frameworks end up being more complementary than competitive among each other. One thing was made clear, in all references, it does not matter if you choose Hadoop, Spark or Storm, having the HDFS is an advantage, because it solves many of storage problems associated with big data computing. So Hadoop is kind of "mandatory", if you need HDFS benefits.

For Spark, its best use cases, are iterative Machine Learning algorithms and Interactive analytics. Furthermore, Spark plus Hadoop is always better than only Hadoop, except when the work dataset size exceeds the individual node RAM size, so in a way it depends on the available infrastructure or required work dataset size.

Storm is a good choice if you need sub-second latency and no data loss. Spark Streaming is better if you need stateful computation, with the guarantee that each event is processed exactly once. Spark Streaming programming logic may also be easier because it's similar to batch programming, in that way, you are working with batches (albeit very small ones)[19].

One key difference between these two technologies is that Spark performs Data-Parallel computations while Storm performs Task-Parallel computations.

4 Conclusion

After this analysis it is possible to realize that the Hadoop framework will stay around for a while, and for a good reason. Even knowing that MapReduce cannot solve every problem, it is still a good choice for research, experimentation, and everyday data manipulation. One of the other frameworks abovementioned, may be better if the advantages of HDFS are not necessarily imperative, or if the use cases are compatible with the framework capabilities, and consequently able to take advantage of its benefits.

In overall, the most suitable platform must always take into account the scenario to witch the system is most focussed.

It's important to acknowledge that the newer Hadoop versions based on YARN, allow Spark to run on top of Hadoop, and there is on-going work to achieve the same with Storm.

It remains to be seen how successful those implementations are, and also how they compare to its native counterparts versions Spark and Storm, since these aspects weren't approached by this survey.

Acknowledgements

The author takes here the chance to say thanks to all the reviewers anonymous that helped whit their revision to improve the resulting quality of this paper.

References

1. Yahoo! Hadoop Tutorial. https://developer.yahoo.com/hadoop/tutorial/. Accessed 20 Dec 2014.
2. Aridhi S (2014) Frameworks for Distributed Computing Sabeur Aridhi.
3. What is Hadoop. http://www-01.ibm.com/software/data/infosphere/hadoop/. Accessed 22 Dec 2014.
4. Rosario R (2011) No Title. http://www.bytemining.com/2011/08/hadoop-fatigue-alternatives-to-hadoop/. Accessed 15 Dec 2014.
5. Welcome to Apache™ Hadoop®! http://hadoop.apache.org/. Accessed 20 Dec 2014.
6. Apache Spark. https://spark.apache.org/. Accessed 26 Dec 2014.
7. Xin R, Rosen J, Zaharia M (2013) Shark: SQL and rich analytics at scale.
8. Zaharia M, Das T, Li H, et al. (2012) Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. Proc. 4th Edition.
9. Apache Storm. https://storm.apache.org/. Accessed 27 Dec 2014.
10. Xuhui Liu; Jizhong Han; Yunqin Zhong; Chengde Han; Xubin He, Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS, Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on , vol., no., pp.1,8, Aug. 31 2009-Sept. 4 2009.
11. L. Jiang, B. Li, M. Song, THE optimization of HDFS based on small files, In 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT2010), Beijing, 2010. pp. 912-915.
12. G. Mackey, S. Sehrish, J. Wang, Improving metadata management for small files in HDFS, In 2009 IEEE International Conference on Cluster Computing and Workshops (CLUSTER'09), New Orleans,Sept, 2009, pp.1-4.
13. Jiong Xie; Shu Yin; Xiaojun Ruan; Zhiyang Ding; Yun Tian; Majors, J.; Manzanares, A.; Xiao Qin, Improving MapReduce performance through data placement in heterogeneous Hadoop clusters, Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on , vol., no., pp.1,9, 19-23 April 2010.

14. Thanh, T.D.; Mohan, S.; Eunmi Choi; SangBum Kim; Pilsung Kim, A Taxonomy and Survey on Distributed File Systems, Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on, vol.1, no., pp.144,149, 2-4 Sept. 2008.
15. S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pages 29–43, New York, NY, USA, 2003. ACM.
16. J. M. Hellerstein, M. Stonebraker, and J. Hamilton. Architecture of a database system. Foundations and Trends in Databases, 1(2): 141–259,2007.
17. Apache Storm vs. Apache Spark. http://www.zdatainc.com/2014/09/apache-storm-apache-spark/. Accessed 20 Dec 2014.
18. Storm vs. Spark Streaming: Side-by-side comparison. http://xinhstechblog.blogspot.pt/2014/06/storm-vs-spark-streaming-side-by-side.html. Accessed 20 Dec 2014.
19. How to run Storm on Apache Mesos. https://mesosphere.com/docs/tutorials/run-storm-on-mesos/. Accessed 20 Dec 2014.
20. Storm on YARN Install on HDP2 Cluster. http://hortonworks.com/kb/storm-on-yarn-install-on-hdp2-beta-cluster/. Accessed 20 Dec 2014.