In The Name of GOD

“Open source big-data solutions”

Hadi Sotudeh (90110618)

Solution	Developer	Type	Description
Hadoop	Apache	Batch	First open source implementation of the MapReduce paradigm
Spark	UC Berkeley AMPLab	Batch	Recent analytics platform that supports in-memory data sets and resiliency
Storm	Twitter	Streaming	Twitter's new streaming big-data analytics solution (new paradigm)
S4	Yahoo	Streaming	Distributed stream computing platform from Yahoo!(map-reduce like paradigm)

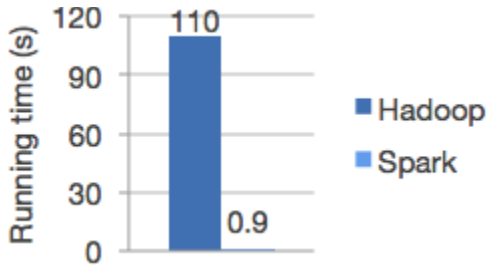
Hadoop:

- The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and thus should be automatically handled in software by the framework.

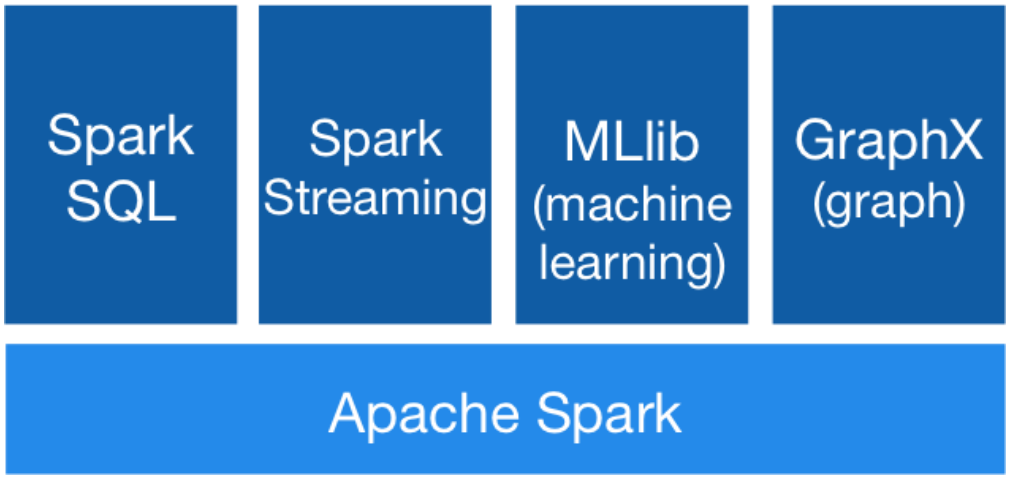
Spark:

- A fast and general engine for large-scale data processing.
- Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk
- Combine SQL, streaming, and complex analytics.
- Powers a stack of high-level tools including Spark SQL, MLlib for machine learning, GraphX, and Spark Streaming.
- Integrated with Hadoop

- Implemented in the Scala language and uses Scala as its application framework but Write applications quickly in Java, Scala or Python.
- Can read from HDFS, HBase, Cassandra, and any Hadoop data source.



Logistic regression in Hadoop and Spark



Storm:

- A free and open source distributed realtime computation system.
- Easy to reliably process unbounded streams of data (realtime processing)
- Use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more.
- Fast: a benchmark clocked it at over a million tuples processed per second per node.
- scalable, fault-tolerant, guarantees your data will be processed.
- Easy to set up and operate.

- language independent.
- Use Cases: Apache Storm: near real-time analytics, natural language processing, data normalization and ETL transformations.
- Standing apart from traditional Map Reduce and other course-grained technologies yielding fine-grained transformations allowing very flexible processing topologies.

Storm vs Hadoop vs Spark

What differentiates Storm from other big-data solutions is the paradigm that it addresses. Hadoop is fundamentally a batch processing system. This model is sufficient for many cases (such as indexing the web). Data is introduced into the Hadoop file system (HDFS) and distributed across nodes for processing. When the processing is complete, the resulting data is returned to HDFS for use by the originator. Storm supports the construction of topologies that transform unterminated streams of data. Those transformations, unlike Hadoop jobs, never stop, instead continuing to process data as it arrives. but other use models exist in which real-time information from highly dynamic sources is required. Solving this problem resulted in the introduction of Storm from Nathan Marz. Storm operates not on static data but on streaming data that is expected to be continuous. With Twitter users generating 140 million tweets per day, it's easy to see how this technology is useful.

Unlike Hadoop, Storm is a computation system and incorporates no concept of storage. This allows Storm to be used in a variety of contexts, whether data arrives dynamically from a nontraditional source or is stored in a storage system such as a database.

Spark is an open source cluster computing environment similar to Hadoop, but it has some useful differences that make it superior in certain workloads—namely, Spark enables in-memory distributed datasets that optimize iterative workloads in addition to interactive queries.

Although Spark was created to support iterative jobs on distributed datasets, it's actually complementary to Hadoop and can run side by side over the Hadoop file system. Spark was developed at the University of California,

Berkeley, Algorithms, Machines, and People Lab to build large-scale and low-latency data analytics applications.

Although Spark has similarities to Hadoop, it represents a new cluster computing framework with useful differences. First, Spark was designed for a specific type of workload in cluster computing—namely, those that reuse a working set of data across parallel operations (such as machine learning algorithms). To optimize for these types of workloads, Spark introduces the concept of in-memory cluster computing, where datasets can be cached in memory to reduce their latency of access.

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.

Regardless though, Storm can certainly still consume files from HDFS and/ or write files to HDFS.

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).

Apache Storm is 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.

Storm and Spark are focused on fairly different use cases. The more "apples-to-apples" comparison would be between Storm and Spark Streaming. Since Spark's RDDs are inherently immutable, Spark Streaming implements a method for "batching" incoming updates in user-defined time intervals that get transformed into their own RDDs. Spark's parallel operators can then

perform computations on these RDDs. This is different from Storm which deals with each event individually.

One key difference between these two technologies is that Spark performs Data-Parallel computationswhile Storm performs Task-Parallel computations. Either design makes tradeoffs that are worth knowing. I would suggest checking out these links.

S4 vs Storm:

Programming model.

S4 implements the Actors programming paradigm. You define your program in terms of Processing Elements (PEs) and Adapters, and the framework instantiates one PE per each unique key in the stream. This means that the logic inside a PE can be very simple, very much like MapReduce.

Storm does not have an explicit programming paradigm. You define your program in terms of bolts and spouts that process partitions of streams. The number of bolts to instantiate is defined a-priori and each bolt will see a partition of the stream.

Data pipeline.

S4 uses a push model, events are pushed to the next PE as fast as possible. If receiver buffers get full events are dropped, and this can happen at any stage in the pipeline (from the Adapter to any PE).

Storm uses a pull model. Each bolt pulls event from its source, be it a spout or another bolt. Event loss can thus happen only at ingestion time, in the spouts if they cannot keep up with the external event rate.

Fault tolerance.

S4 provides state recovery via uncoordinated checkpointing. When a node crashes, a new node takes over its task and restarts from a recent snapshot of its state. Events sent after the last checkpoint and before the recovery are lost. Indeed, events can be lost in any case due to overload, so this design makes perfect sense. State recovery is very important for long running machine learning programs, where the state represents days or weeks worth of data.

Storm provides guaranteed delivery of events/tuples. Each tuple traverses the entire pipeline within a time interval or is declared as failed and can be replayed from the start by the spout. Spouts are responsible to keep tuples around for replay, the framework provides no state recovery.

S4:

- Clean programming model.
- State recovery.
- Inter-app communication.

Storm:

- Guaranteed processing.
- More mature, more traction, larger community.
- High performance.
- Advanced features (thread programming support, transactional topologies).

Useful Links:

1- http://www.datasalt.com/2012/01/real-time-feed-processing-with-storm/

2- http://www.ibm.com/developerworks/library/os-twitterstorm/

3- http://spark.apache.org

4- http://storm.apache.org