Spark

Hadoop is parallel data processing framework that has traditionally been  used to run map/reduce jobs. These are long running batch jobs that  take minutes or hours to complete. Spark is an alternative to the  traditional batch map/reduce model that can be used for real-time stream  data processing and fast interactive queries that finish within  seconds. It is also based on the Hadoop framework. So, Hadoop is  evolving into a general purpose framework that supports multiple models,  such as traditional map/reduce and Spark

Comparision of MapReduce and Spark

**SPARK**, apt choice for the future big data applications that possibly would require lower latency queries, iterative computation and real time processing on data.  
  
**i) Faster**  
  
Execute batch processing jobs , about 10 to 100 times faster than the Hadoop MapReduce framework just by merely cutting down on the number of reads and writes to the disc.   
  
MapReduce does not leverage the memory of the Hadoop cluster to the maximum. In Spark the concept of RDDs (Resilient Distributed Datasets) lets you save data on memory and preserve it to the disc if and only if it is required and as well it does not have any kind of synchronization barriers that possibly could slow down the process. Thus the general execution engine of Spark is much faster than Hadoop MapReduce with the use of memory.  
  
**ii) Easy Management**  
  
With Hadoop Spark  it is possible to perform Streaming, Batch Processing and Machine Learning all in the same cluster.   
  
With Spark it is possible to control different kinds of workloads, so if there is an interaction between various workloads in the same process it is easier to manage and secure such workloads which come as a limitation with MapReduce.  
  
**iii) Spark Streaming –Real Time Method to Process Streams**  
  
In case of Hadoop MapReduce you just get to process a batch of stored data but with Hadoop Spark it is as well possible to modify the data in real time through Spark Streaming.   
  
spark streaming is based on a paper Discretized Streams, which proposes a new model for doing windowed computations on streams using micro batches. Hadoop doesn't support  this.  
  
**iv) Caching**  
  
Spark ensures lower latency computations by caching the partial results across its memory of distributed workers unlike MapReduce which is disk oriented completely.   
  
**V) Recovery**  
  
RDD is the main abstraction of spark. It allows recovery of failed nodes by re-computation of the DAG while also supporting a more similar recovery style to Hadoop by way of checkpointing, to reduce the dependencies of an RDD. Storing a spark job in a DAG allows for lazy computation of RDD's and can also allow spark's optimization engine to schedule the flow in ways that make a big difference in performance.  
  
**vi) Spark API**  
  
Hadoop MapReduce has a very strict API that doesn't allow for as much versatility. Since spark abstracts away many of the low level details it allows for more productivity. Also things like broadcast variables and accumulators are much more versatile than DistributedCache and counters IMO.  
  
  
**Vii) Scheduler**  
  
As a product of in memory computation spark sort of acts as it's own flow scheduler. Whereas with standard MR you need an external job scheduler like Azkaban or Oozie to schedule complex flows.  
  
**Viii)  Iterative computations**  
  
Spark has the upper hand as long as we’re talking about iterative computations that need to pass over the same data many times. But when it comes to one-pass ETL-like jobs, for example, data transformation or data integration, then MapReduce is the deal—this is what it was designed for.  
  
**ix) Cost**  
  
The memory in the Spark cluster should be at least as large as the amount of data you need to process, because the data has to fit into the memory for optimal performance. So, if you need to process really Big Data, Hadoop will definitely be the cheaper option since hard disk space comes at a much lower rate than memory space.  
  
On the other hand, considering Spark’s benchmarks, it should be more cost-effective since less hardware can perform the same tasks much faster, especially on the cloud where compute power is paid per use.  
  
**x) Failure Tolerance**  
  
  
Spark has retries per task and speculative execution—just like MapReduce.   
  
If a process crashes in the middle of execution, it could continue where it left off, whereas Spark will have to start processing from the beginning.   
  
**xi) Security**  
  
Spark security is still in its infancy; Hadoop MapReduce has more security features and projects like kerborose , sentry etc .

RDDs achieve fault tolerance through a notion of *lineage:*if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to be able to rebuild just that partition." This removes the need for replication to achieve fault tolerance. Hadoop, on the other hand, uses replication to achieve fault tolerance

**Spark’s major use cases over Hadoop**

* Iterative Algorithms in Machine Learning
* Interactive Data Mining and Data Processing
* Spark is a fully Apache Hive-compatible data warehousing system that can run 100x faster than Hive.
* Stream processing: Log processing and Fraud detection in live streams for alerts, aggregates and analysis
* Sensor data processing: Where data is fetched and joined from multiple sources, in-memory dataset really helpful as they are easy and fast to process.

