**->O que é o MapReduce (explicar conceitos)**

**->Para que é usado?**

**->Exemplos de aplicação**

**->Quem usa?**

**->Distribuiçoes populares**

**http://en.wikipedia.org/wiki/Free\_monoid**

**MapReduce** is a [programming model](http://en.wikipedia.org/wiki/Programming_model) and an associated implementation for processing and generating large data sets with a [parallel](http://en.wikipedia.org/wiki/Parallel_computing), [distributed](http://en.wikipedia.org/wiki/Distributed_computing) algorithm on a [cluster](http://en.wikipedia.org/wiki/Cluster_(computing)).[[1]](http://en.wikipedia.org/wiki/MapReduce#cite_note-1)[[2]](http://en.wikipedia.org/wiki/MapReduce#cite_note-2)

A MapReduce program is composed of a **Map()** [procedure](http://en.wikipedia.org/wiki/Procedure_(computing)) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a **Reduce()** procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by [marshalling](http://en.wikipedia.org/wiki/Marshalling_(computer_science)) the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for [redundancy](http://en.wikipedia.org/wiki/Redundancy_(engineering)) and [fault tolerance](http://en.wikipedia.org/wiki/Fault-tolerant_computer_system).

The model is inspired by the [map](http://en.wikipedia.org/wiki/Map_(higher-order_function)) and [reduce](http://en.wikipedia.org/wiki/Fold_(higher-order_function)) functions commonly used in [functional programming](http://en.wikipedia.org/wiki/Functional_programming),[[3]](http://en.wikipedia.org/wiki/MapReduce#cite_note-map-3) although their purpose in the MapReduce framework is not the same as in their original forms.[[4]](http://en.wikipedia.org/wiki/MapReduce#cite_note-4) The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once. As such, a [single-threaded](http://en.wikipedia.org/wiki/Single-threaded) implementation of MapReduce (such as [MongoDB](http://en.wikipedia.org/wiki/MongoDB)) will usually not be faster than a traditional (non-MapReduce) implementation, any gains are usually only seen with [multi-threaded](http://en.wikipedia.org/wiki/Multi-threaded) implementations.[[5]](http://en.wikipedia.org/wiki/MapReduce#cite_note-stackoverflow-5) Only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play, is the use of this model beneficial.

Many tasks: Process lots of data to produce other data

Want to use hundreds or thousands of CPUs

* ... but this needs to be easy

MapReduce provides:

* Automatic parallelization and distribution
* Fault-tolerance
* I/O scheduling
* Status and monitoring

The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

MapReduce is emerging as an important programming model for large-scale data-parallel applications such as web indexing, data mining, and scientific simulation. Hadoop is an open-source implementation of MapReduce enjoying wide adoption and is often used for short jobs where low response time is critical. Hadoop's performance is closely tied to its task scheduler, which implicitly assumes that cluster nodes are homogeneous and tasks make progress linearly, and uses these assumptions to decide when to speculatively re-execute tasks that appear to be stragglers. In practice, the homogeneity assumptions do not always hold. An especially compelling setting where this occurs is a virtualized data center, such as Amazon's Elastic Compute Cloud (EC2). We show that Hadoop's scheduler can cause severe performance degradation in heterogeneous environments. We design a new scheduling algorithm, Longest Approximate Time to End (LATE), that is highly robust to heterogeneity. LATE can improve Hadoop response times by a factor of 2 in clusters of 200 virtual machines on EC2.

The MapReduce model popularized by Google is very attractive for ad-hoc parallel processing of arbitrary data. MapReduce breaks a computation into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. Its popular open-source implementation, Hadoop [[2](http://static.usenix.org/legacy/events/osdi08/tech/full_papers/zaharia/zaharia_html/" \l "hadoop)], was developed primarily by Yahoo, where it runs jobs that produce hundreds of terabytes of data on at least 10,000 cores [[4](http://static.usenix.org/legacy/events/osdi08/tech/full_papers/zaharia/zaharia_html/" \l "yahoocluster)]. Hadoop is also used at Facebook, Amazon, and Last.fm [[5](http://static.usenix.org/legacy/events/osdi08/tech/full_papers/zaharia/zaharia_html/" \l "poweredbyhadoop)]. In addition, researchers at Cornell, Carnegie Mellon, University of Maryland and PARC are starting to use Hadoop for seismic simulation, natural language processing, and mining web data [[5](http://static.usenix.org/legacy/events/osdi08/tech/full_papers/zaharia/zaharia_html/#poweredbyhadoop),[6](http://static.usenix.org/legacy/events/osdi08/tech/full_papers/zaharia/zaharia_html/" \l "hadoop-summit)].

STEPS

MapReduce is a framework for processing [parallelizable](http://en.wikipedia.org/wiki/Parallel_computing) problems across huge datasets using a large number of computers (nodes), collectively referred to as a [cluster](http://en.wikipedia.org/wiki/Computer_cluster) (if all nodes are on the same local network and use similar hardware) or a [grid](http://en.wikipedia.org/wiki/Grid_Computing) (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Processing can occur on data stored either in a [filesystem](http://en.wikipedia.org/wiki/Filesystem) (unstructured) or in a [database](http://en.wikipedia.org/wiki/Database) (structured). MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.

* **"Map" step:** Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.
* **"Shuffle" step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
* **"Reduce" step:** Worker nodes now process each group of output data, per key, in parallel.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is [associative](http://en.wikipedia.org/wiki/Associative_property). While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large [server farm](http://en.wikipedia.org/wiki/Server_farm) can use MapReduce to sort a [petabyte](http://en.wikipedia.org/wiki/Petabyte) of data in only a few hours.[[6]](http://en.wikipedia.org/wiki/MapReduce#cite_note-6) The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

1. **Prepare the Map() input** – the "MapReduce system" designates Map processors, assigns the input key value *K1* that each processor would work on, and provides that processor with all the input data associated with that key value.
2. **Run the user-provided Map() code** – Map() is run exactly once for each *K1* key value, generating output organized by key values *K2*.
3. **"Shuffle" the Map output to the Reduce processors** – the MapReduce system designates Reduce processors, assigns the *K2* key value each processor should work on, and provides that processor with all the Map-generated data associated with that key value.
4. **Run the user-provided Reduce() code** – Reduce() is run exactly once for each *K2* key value produced by the Map step.
5. **Produce the final output** – the MapReduce system collects all the Reduce output, and sorts it by *K2* to produce the final outcome.

These five steps can be Logically thought of as running in sequence – each step starts only after the previous step is completed – although in practice they can be interleaved as long as the final result is not affected.

In many situations, the input data might already be distributed (["sharded"](http://en.wikipedia.org/wiki/Shard_(database_architecture))) among many different servers, in which case step 1 could sometimes be greatly simplified by assigning Map servers that would process the locally present input data. Similarly, step 3 could sometimes be sped up by assigning Reduce processors that are as close as possible to the Map-generated data they need to process.

MapReduce is useful in a wide range of applications, including distributed pattern-based searching, distributed sorting, web link-graph reversal, Singular Value Decomposition,[[9]](http://en.wikipedia.org/wiki/MapReduce#cite_note-9)web access log stats, [inverted index](http://en.wikipedia.org/wiki/Inverted_index) construction, [document clustering](http://en.wikipedia.org/wiki/Document_clustering), [machine learning](http://en.wikipedia.org/wiki/Machine_learning),[[10]](http://en.wikipedia.org/wiki/MapReduce#cite_note-mrml-10) and [statistical machine translation](http://en.wikipedia.org/wiki/Statistical_machine_translation). Moreover, the MapReduce model has been adapted to several computing environments like multi-core and many-core systems,[[11]](http://en.wikipedia.org/wiki/MapReduce#cite_note-evalMR-11)[[12]](http://en.wikipedia.org/wiki/MapReduce#cite_note-graphicsMR-12)[[13]](http://en.wikipedia.org/wiki/MapReduce#cite_note-tiledMR-13) desktop grids,[[14]](http://en.wikipedia.org/wiki/MapReduce#cite_note-gridMR-14) volunteer computing environments,[[15]](http://en.wikipedia.org/wiki/MapReduce#cite_note-volunteerMR-15) dynamic cloud environments,[[16]](http://en.wikipedia.org/wiki/MapReduce#cite_note-dynCloudMR-16) and mobile environments.[[17]](http://en.wikipedia.org/wiki/MapReduce#cite_note-mobileMR-17)

At Google, MapReduce was used to completely regenerate Google's index of the [World Wide Web](http://en.wikipedia.org/wiki/World_Wide_Web). It replaced the old *ad hoc* programs that updated the index and ran the various analyses.[[18]](http://en.wikipedia.org/wiki/MapReduce#cite_note-usage-18) Development at Google has since moved on to technologies such as Percolator, Flume and MillWheel that offer streaming operation and updates instead of batch processing, to allow integrating "live" search results without rebuilding the complete index.

MapReduce's stable inputs and outputs are usually stored in a [distributed file system](http://en.wikipedia.org/wiki/Distributed_file_system). The transient data is usually stored on local disk and fetched remotely by the reducers.

Apache Hadoop

**Apache Hadoop** is an [open-source](http://en.wikipedia.org/wiki/Open_source) [software framework](http://en.wikipedia.org/wiki/Software_framework) for [distributed storage](http://en.wikipedia.org/wiki/Clustered_file_system) and [distributed processing](http://en.wikipedia.org/wiki/Distributed_processing) of [Big Data](http://en.wikipedia.org/wiki/Big_Data) on [clusters](http://en.wikipedia.org/wiki/Computer_cluster) of[commodity hardware](http://en.wikipedia.org/wiki/Commodity_hardware). Its [Hadoop Distributed File System (HDFS)](http://en.wikipedia.org/wiki/Apache_Hadoop#HDFS) splits files into large blocks (default 64MB or 128MB) and distributes the blocks amongst the nodes in the cluster. For processing the data, the Hadoop [Map/Reduce](http://en.wikipedia.org/wiki/MapReduce) ships code (specifically[Jar files](http://en.wikipedia.org/wiki/Jar_files)) to the nodes that have the required data, and the nodes then process the data in parallel. This approach takes advantage of data locality,[[3]](http://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-3) in contrast to conventional [HPC architecture](http://en.wikipedia.org/wiki/Supercomputer_architecture) which usually relies on a [parallel file system](http://en.wikipedia.org/wiki/Parallel_file_system) (compute and data separated, but connected with high-speed networking)

[Spark](https://spark.incubator.apache.org/)

* Authored by [AMPLab at UC Berkeley](https://amplab.cs.berkeley.edu/), now an Apache Incubator project
* "Next-generation" map/reduce; designed for performance
* APIs for Scala, Java, and Python
* Includes Shark (SQL), streaming API, MLlib (machine learning)

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala and Python, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including [Spark SQL](http://spark.apache.org/docs/latest/sql-programming-guide.html) for SQL and structured data processing, [MLlib](http://spark.apache.org/docs/latest/mllib-guide.html) for machine learning, [GraphX](http://spark.apache.org/docs/latest/graphx-programming-guide.html) for graph processing, and [Spark Streaming](http://spark.apache.org/docs/latest/streaming-programming-guide.html).

The input data takes the form of a file that contains key/value pairs. Users specify a map function that iterates over this input file and generates, for each key/value pair, a set of intermediate key/value pairs. For this, the map function must parse the value field associated with each key to extract any required attributes. Users also specify a reduce function that, similar to a relational aggregate operator, merges or aggregates all values associated with the same key. MapReduce jobs are automatically parallelized and executed on a cluster of commodity machines: the map stage is partitioned into multiple map tasks and the reduce stage is partitioned into multiple reduce tasks. Each map task reads and processes a distinct chunk of the partitioned and distributed input data. The degree of parallelism in the map stage depends on the input data size. The output of the map stage is hash partitioned across a configurable number of reduce tasks. Data between the map and reduce stages is always materialized. As discussed below, a higher-level query may require multiple MapReduce jobs, each of which has map tasks followed by reduce tasks. Data between consecutive jobs is also always materialized.