An introduction to Apache Hadoop for big data

Apache Hadoop is an open source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users. It is licensed under the Apache License 2.0.

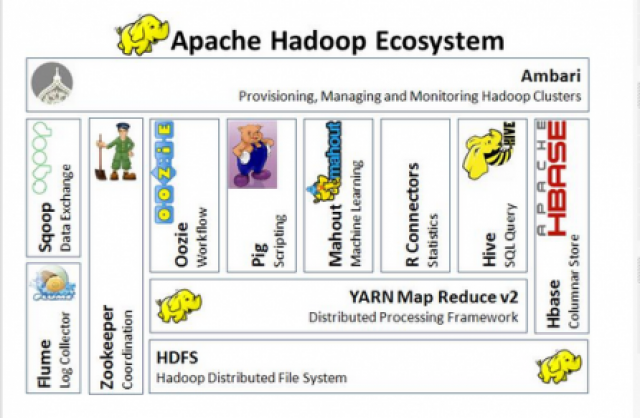
Hadoop was created by Doug Cutting and Mike Cafarella in 2005. It was originally developed to support distribution for the [Nutch search engine](http://nutch.apache.org/) project. Doug, who was working at Yahoo! at the time and is now Chief Architect of [Cloudera](http://www.cloudera.com/), named the project after his son's toy elephant. Cutting's son was 2 years old at the time and just beginning to talk. He called his beloved stuffed yellow elephant "Hadoop" (with the stress on the first syllable). Now 12, Doug's son often exclaims, "Why don't you say my name, and why don't I get royalties? I deserve to be famous for this!"

**The Apache Hadoop framework is composed of the following modules**

1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules
2. Hadoop Distributed File System (HDFS): a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster
3. Hadoop YARN: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications
4. Hadoop MapReduce: a programming model for large scale data processing

All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers.

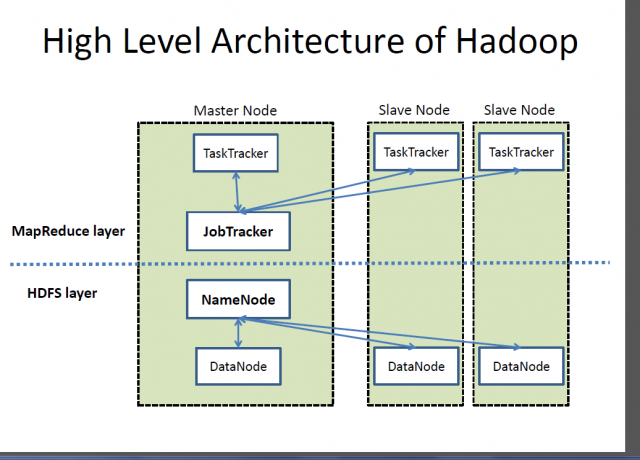
Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop "platform" is now commonly considered to consist of a number of related projects as well: Apache Pig, Apache Hive, Apache HBase, and others.



For the end-users, though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig and Apache Hive, among other related projects, expose higher level user interfaces like Pig latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts.

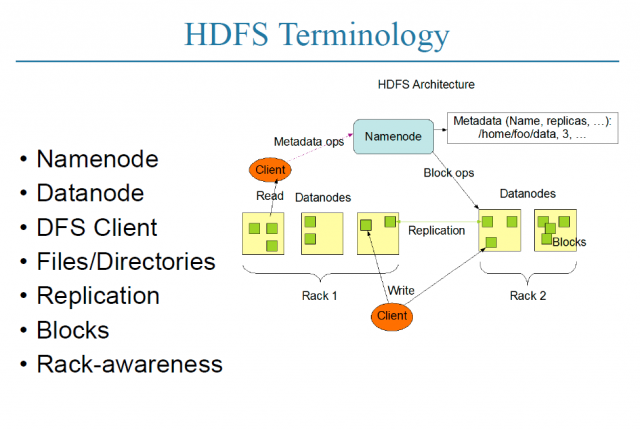
**HDFS and MapReduce**

There are two primary components at the core of Apache Hadoop 1.x: the Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework. These are both open source projects, inspired by technologies created inside Google.



**Hadoop distributed file system**

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. Each node in a Hadoop instance typically has a single namenode, and a cluster of datanodes form the HDFS cluster. The situation is typical because each node does not require a datanode to be present. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses the TCP/IP layer for communication. Clients use Remote procedure call (RPC) to communicate between each other.



HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high. HDFS is not fully POSIX-compliant, because the requirements for a POSIX file-system differ from the target goals for a Hadoop application. The tradeoff of not having a fully POSIX-compliant file-system is increased performance for data throughput and support for non-POSIX operations such as Append.

HDFS added the high-availability capabilities for release 2.x, allowing the main metadata server (the NameNode) to be failed over manually to a backup in the event of failure, automatic fail-over.

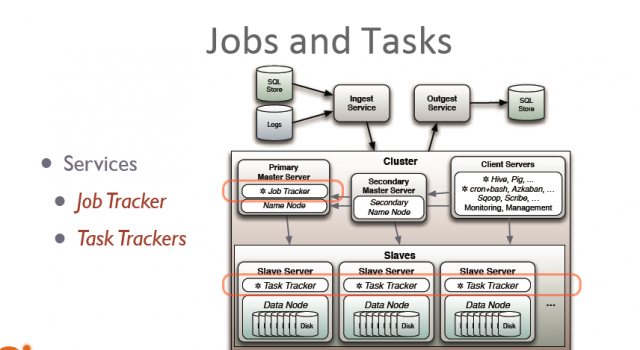
The HDFS file system includes a so-called secondary namenode, which misleads some people into thinking that when the primary namenode goes offline, the secondary namenode takes over. In fact, the secondary namenode regularly connects with the primary namenode and builds snapshots of the primary namenode's directory information, which the system then saves to local or remote directories. These checkpointed images can be used to restart a failed primary namenode without having to replay the entire journal of file-system actions, then to edit the log to create an up-to-date directory structure. Because the namenode is the single point for storage and management of metadata, it can become a bottleneck for supporting a huge number of files, especially a large number of small files. HDFS Federation, a new addition, aims to tackle this problem to a certain extent by allowing multiple name-spaces served by separate namenodes.

An advantage of using HDFS is data awareness between the job tracker and task tracker. The job tracker schedules map or reduce jobs to task trackers with an awareness of the data location. For example, if node A contains data (x, y, z) and node B contains data (a, b, c), the job tracker schedules node B to perform map or reduce tasks on (a,b,c) and node A would be scheduled to perform map or reduce tasks on (x,y,z). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer. When Hadoop is used with other file systems, this advantage is not always available. This can have a significant impact on job-completion times, which has been demonstrated when running data-intensive jobs. HDFS was designed for mostly immutable files and may not be suitable for systems requiring concurrent write-operations.

Another limitation of HDFS is that it cannot be mounted directly by an existing operating system. Getting data into and out of the HDFS file system, an action that often needs to be performed before and after executing a job, can be inconvenient. A filesystem in Userspace (FUSE) virtual file system has been developed to address this problem, at least for Linux and some other Unix systems.

File access can be achieved through the native Java API, the Thrift API, to generate a client in the language of the users' choosing (C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk, or OCaml), the command-line interface, or browsed through the HDFS-UI web app over HTTP.

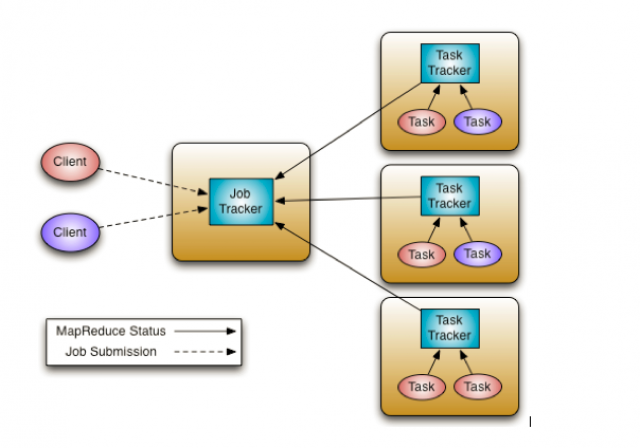
**JobTracker and TaskTracker: The MapReduce engine**



Above the file systems comes the MapReduce engine, which consists of one JobTracker, to which client applications submit MapReduce jobs. The JobTracker pushes work out to available TaskTracker nodes in the cluster, striving to keep the work as close to the data as possible.

With a rack-aware file system, the JobTracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network.

If a TaskTracker fails or times out, that part of the job is rescheduled. The TaskTracker on each node spawns off a separate Java Virtual Machine process to prevent the TaskTracker itself from failing if the running job crashes the JVM. A heartbeat is sent from the TaskTracker to the JobTracker every few minutes to check its status. The Job Tracker and TaskTracker status and information is exposed by Jetty and can be viewed from a web browser.



If the JobTracker failed on Hadoop 0.20 or earlier, all ongoing work was lost. Hadoop version 0.21 added some checkpointing to this process. The JobTracker records what it is up to in the file system. When a JobTracker starts up, it looks for any such data, so that it can restart work from where it left off.

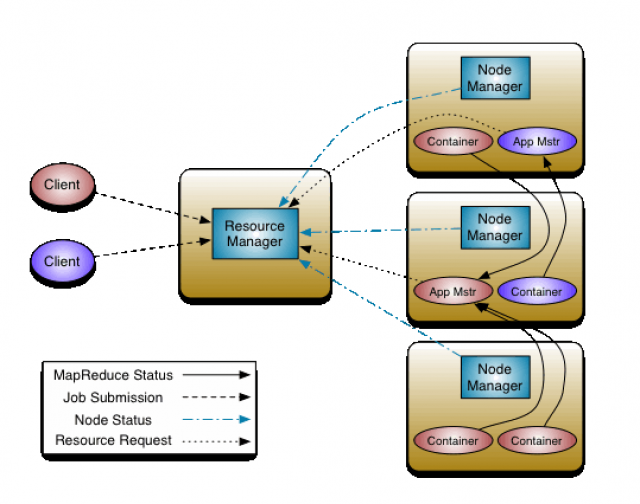
**Known limitations of this approach in Hadoop 1.x**

The allocation of work to TaskTrackers is very simple. Every TaskTracker has a number of available slots (such as "4 slots"). Every active map or reduce task takes up one slot. The Job Tracker allocates work to the tracker nearest to the data with an available slot. There is no consideration of the current system load of the allocated machine, and hence its actual availability. If one TaskTracker is very slow, it can delay the entire MapReduce job—especially towards the end of a job, where everything can end up waiting for the slowest task. With speculative execution enabled, however, a single task can be executed on multiple slave nodes.

**Apache Hadoop NextGen MapReduce (YARN)**

MapReduce has undergone a complete overhaul in hadoop-0.23 and we now have, what we call, MapReduce 2.0 (MRv2) or YARN.

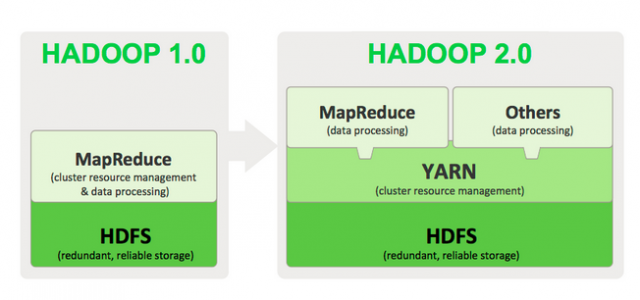
Apache™ Hadoop® YARN is a sub-project of Hadoop at the Apache Software Foundation introduced in Hadoop 2.0 that separates the resource management and processing components. YARN was born of a need to enable a broader array of interaction patterns for data stored in HDFS beyond MapReduce. The YARN-based architecture of Hadoop 2.0 provides a more general processing platform that is not constrained to MapReduce.



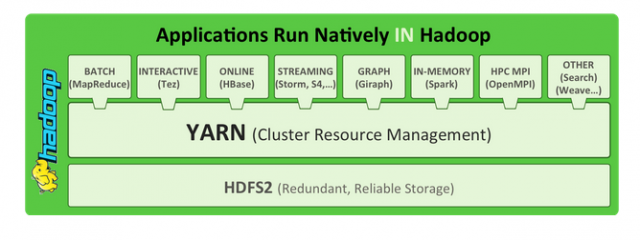
The fundamental idea of MRv2 is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.

The ResourceManager and per-node slave, the NodeManager (NM), form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.



As part of Hadoop 2.0, YARN takes the resource management capabilities that were in MapReduce and packages them so they can be used by new engines. This also streamlines MapReduce to do what it does best, process data. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource management. Many organizations are already building applications on YARN in order to bring them IN to Hadoop.



As part of Hadoop 2.0, YARN takes the resource management capabilities that were in MapReduce and packages them so they can be used by new engines. This also streamlines MapReduce to do what it does best, process data. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource management. Many organizations are already building applications on YARN in order to bring them IN to Hadoop. When enterprise data is made available in HDFS, it is important to have multiple ways to process that data. With Hadoop 2.0 and YARN organizations can use Hadoop for streaming, interactive and a world of other Hadoop based applications.

**What YARN does**

YARN enhances the power of a Hadoop compute cluster in the following ways:

* Scalability: The processing power in data centers continues to grow quickly. Because YARN ResourceManager focuses exclusively on scheduling, it can manage those larger clusters much more easily.
* Compatibility with MapReduce: Existing MapReduce applications and users can run on top of YARN without disruption to their existing processes.
* Improved cluster utilization: The ResourceManager is a pure scheduler that optimizes cluster utilization according to criteria such as capacity guarantees, fairness, and SLAs. Also, unlike before, there are no named map and reduce slots, which helps to better utilize cluster resources.
* Support for workloads other than MapReduce: Additional programming models such as graph processing and iterative modeling are now possible for data processing. These added models allow enterprises to realize near real-time processing and increased ROI on their Hadoop investments.
* Agility: With MapReduce becoming a user-land library, it can evolve independently of the underlying resource manager layer and in a much more agile manner.

**How YARN works**

The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker/TaskTracker into separate entities:

* a global ResourceManager
* a per-application ApplicationMaster
* a per-node slave NodeManager and
* a per-application container running on a NodeManager

The ResourceManager and the NodeManager form the new, and generic, system for managing applications in a distributed manner. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The per-application ApplicationMaster is a framework-specific entity and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the component tasks. The ResourceManager has a scheduler, which is responsible for allocating resources to the various running applications, according to constraints such as queue capacities, user-limits etc. The scheduler performs its scheduling function based on the resource requirements of the applications. The NodeManager is the per-machine slave, which is responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager. Each ApplicationMaster has the responsibility of negotiating appropriate resource containers from the scheduler, tracking their status, and monitoring their progress. From the system perspective, the ApplicationMaster runs as a normal container.