Week 7

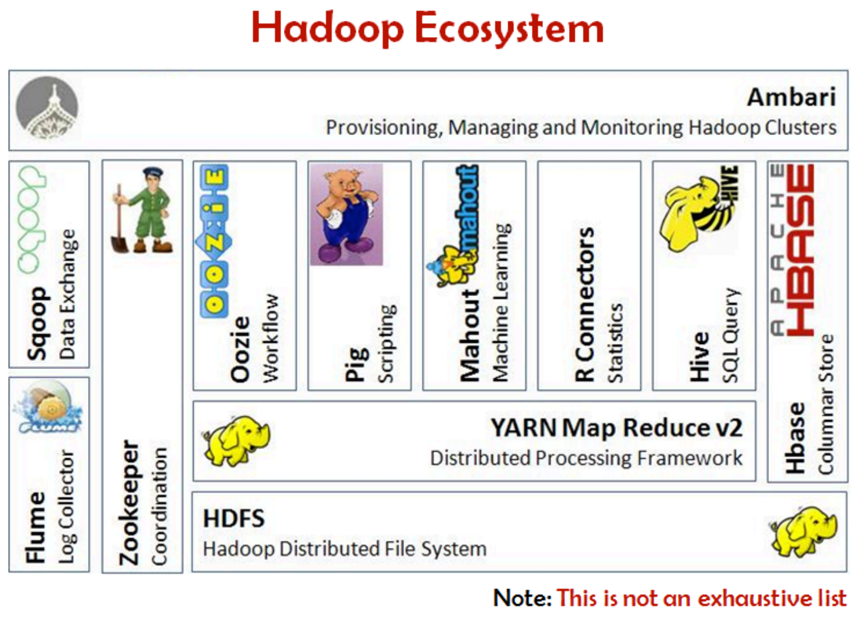
From the Expert: Introduction to Hadoop (ecosystem)

In this data-driven era, many enterprises generate enormous data volumes at the rate of terabyte (10^{12} bytes) or even petabyte (10^{15} bytes) per day. The data volumes tend to rise steeply. There are attempts to search for new approaches for storing and processing data that were once too expensive to retain and was once believed to be of little value. Especially, corporations strive to develop the intelligent systems to comprehend the world (i.e. customers) using this massive collected data. As a result, the fundamentals of storing, managing and processing the data have been shifted. The Hadoop platform has been invented as a cost-effective, and scalable infrastructure for this purpose.

Hadoop Ecosystem

Hadoop is an open source project for processing huge volumes of data. It, along with many related ecosystem components, is free software available from the Apache Foundation (http://hadoop.apache.org/). The Hadoop system has the ability to store, analyze and access large volumes of data quickly and cost effectively. Here are some of the popular projects under the Apache Software Foundation. For the up to date information, please visit the Apache web site.

- Hadoop Mapreduce and HDFS
- Pig
- Hive
- Flume
- Sqoop
- HBase
- Oozie



Source: http://pramodgampa.blogspot.com/2013/07/hadoop-ecosystem.html

Core Technologies of Hadoop
Hadoop (http://hadoop.apache.org/) is based on the Google File System (GFS), which was introduced in 2003 and MapReduce in 2004. Hadoop consists of two core components:

- The Hadoop Distributed File System (HDFS)
- MapReduce

A set of machines running HDFS and Mapreduce is known as Hadoop cluster. A cluster can contain a single node to several thousand nodes.

The Hadoop Distributed File System (HDFS)
The main responsibility of HDFS is to store (massive) data on the cluster. Typically, data is split into blocks and distributed through multiple nodes in the cluster. The size of one block is typically 64 MB or 128 MB. Each block is replicated multiple times and stored on different nodes to ensure reliability and availability. The default for replication is 3 times. This simply means that, each block exists on three different machines. A block is simply chunks of a file or a collection of binary data.

The block size of HDFS is larger that the traditional file systems (4 or 8 KB) so that larger data can be read and written. The times for disk operations will be minimized. As a result, the better performance can be achieved in the streaming I/O operations. Typically, HDFS files are written once. It is not possible to do anything after the replica is written in order to ensure consistency. Likewise, replicas also support reliability, and availability especially in the machine failure situation, as well as locality (data can be read from the closet machine). Here, the replica is a copy of HDFS block. In addition, HDFS also has the mechanism to track and manage the number of replicas. If the number of copies falls below the configured number, a new copy would be automatically created.

Filesystem Management:
To keep track of the information regarding to which blocks make up a file and where those blocks located are the responsibility of a master node called *Namenode*. The information is known as metadata, which is stored in RAM for fast access. Without this metadata, it is impossible to access files in the HDFS cluster.

The Namenode process must be running at all times, otherwise the cluster becomes inaccessible. The Namenode also keeps track of changes on disk for crash recovery. Another process is known as *Secondary Namenode*, which performs housekeeping (i.e. checkpoints) tasks for the primary Namenode. It is important to note that the Secondary Name Node is not a back up for Namenode.

An alternative way to set up for higher availability is that HDFS has been evolved using two Namenodes, which are Active and Standby, instead of a single Namenode. The standby Namenode can automatically take over once the active Name Node fails.

The actual data blocks (64 or 128 MB size per block) are contained by the *Datanode*, which report their blocks and status back to the Namenode. If a Datanode does not report status after a certain period of time, the Namenode will mark it as failed and direct other Datanodes to obtain copies of the data to ensure proper redundancy.

Read the files:
The applications that need to access the files will communicate with the Namenode to find out which blocks make up the files and which Datanodes those blocks are on. After that, the applications will communicate with Datanodes directly to read the data. Files in HDFS are written once and not allowed to randomly written, although they can be appended. HDFS perform well with large and streaming file reading rather than random reads.

Mapreduce
This is a system for processing data in the Hadoop cluster by distributing the task across multiple nodes. Generally, the process is comprised of 2 phases: Map and Reduce. There may also be step between these two phases known as shuffle and sort. MapReduce features include distribution and parallelization, fault tolerance, tools for status and monitoring, clean abstraction (programmers need only focus on Map and Reduce programming and not implementation details). Typically, a MapReduce application is written in Java. But it can also be written using Hadoop streaming; a utility from Hadoop distribution system that allow you to uses any executable code or script to run MapReduce jobs. There are also domain-specific scripting languages, namely Pig and Hive that simplify the task of building MapReduce programs without needing to understand Java.

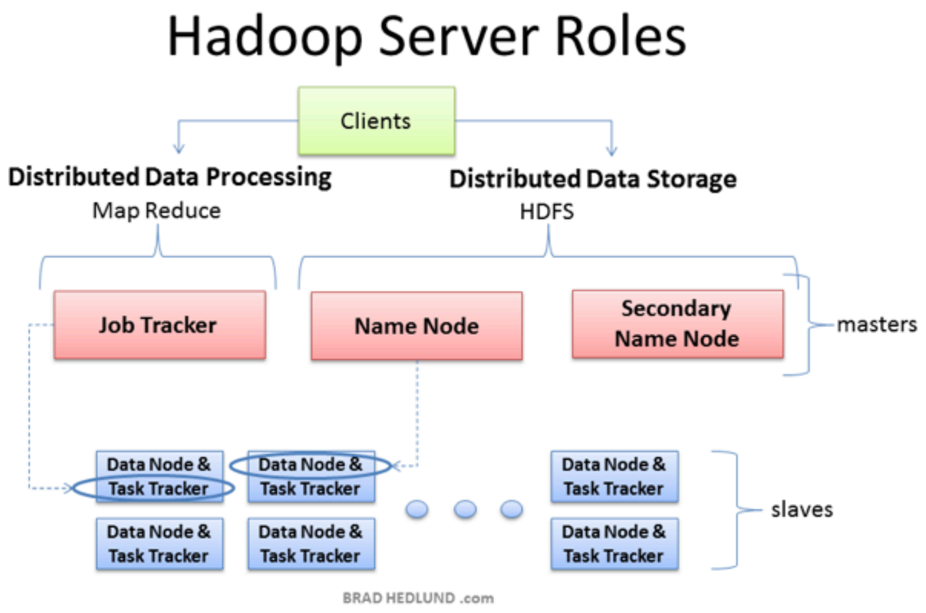
The *JobTracker* is a software process that controls MapReduce jobs and is located on the master node. Typically, the MapReduce jobs are submitted to the JobTracker. Then, the JobTracker distributes Map and Reduce tasks to other nodes in the cluster. The software process called *TaskTracker* on each of these cluster nodes is responsible for instantiating Map and Reduce tasks during their respective phases, as well as, reporting the progress to the JobTracker. A task is the execution on the slice of data of a single Mapper or Reducer. An instance that attempts to execute a task is known as a *task attempt*. In the case of task attempt failure, JobTracker will start another attempt. It is important to note that each node in the cluster that executes the TaskTracker will also be a Datanode, so that tasks can be preferentially started on the node that also has the data it will use. If that is not possible, the data will transfer over the network from the nearest possible Datanode that contains the data.

Mapreduce: The Mapper
Each Mapper processes a portion of a data and run in parallel. The Mapper reads data in the form of key-value pairs and outputs none or more key-value pairs. These outputs are known as intermediate keys-values pairs. After the Map phase, the intermediate values associated with a particular intermediate key are combined into a list. This list is passed to the Reducer.

Mapreduce: The Reducer
The Reducer cannot start until all Mappers are finished. The number of Reducer (single or multiple) is specified in job configuration. The lists that passed to the Reducers are sorted using the key order in the shuffle and sort step. All values related to the same intermediate keys are processed using the same Reducer.
The Reducer outputs none or more key-value pairs and these will be written to HDFS. Principally, the Reducer produces a single key-value pair for each input key.

In summary, the nodes can be categorized into:

- Master Nodes, which run NameNode, Secondary NameNode and JobTracker daemon. In large clusters they will typically located on their own host, but in smaller clusters these daemons can all be in one machine. Each daemon runs in separate Java Virtual Machine.
- Slave Nodes, which run the DataNode and TaskTracker daemons. Each slave node typically runs both of these daemons.



source:http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

The streaming API
The streaming API allows developers to use others languages other than Java to write Mappers and Reducers. The only requirement is that the language should be able to read from standard input and write to standard output. Therefore, there is no need to learn Java and ability to use existing libraries codes. Some concerns regarding to streaming API include: performance, primary text data, excessive usage of RAM, other parts still have to write in Java.

Hive and Pig:
As mentioned earlier, MapReduce code is implemented in Java, which require amounts of effort to understand the programs, think in term of Mapreduce, including write, test and update the code. These, however, are not practical for people who are involved in analyzing the data such as statistician, business analysts, data analysts, etc. Therefore, a high level abstraction on top of MapReduce is mandatory. Hive and Pig can offer the capability to query data without needing to be familiar with MapReduce.

Hive (http://hive.apache.org) is an abstraction on top of MapReduce. Initially, Hive was developed at Facebook. Hive utilizes HiveQL language, which is similar to SQL. Users can query data on Hadoop without knowing Java or MapReduce. The Hive interpreter runs on a client machine and turns HiveQL queries into Mapreduce jobs. Hive facilitates data summarization, ad hoc queries, and analysis of large datasets.

Table definitions are layered on top of data in HDFS. In other words, a table-like schema is defined over the existing files in HDFS. The data on disk is never changed but parsed at the query time. Mapreduce functions corresponding to the SQL statements are assembled and built into an execution plan. Hive tables can contain various column types such as int, float, string, Boolean, array, struct, and map. Table definitions and other metadata are store in a database called *Metastore*, which is stored locally in the client machine by default. However, a Metastore server can be hosted within the cluster if several people operate on Hive and need to share the same data.

Actual data is stored in flat files such as character-delimited text, or SequenceFiles. Nevertheless, Hive does not support all standard SQL command such as UPDATE, DELETE, and no correlated subqueries.

Pig (http://pig.apache.org) is another alternative abstraction on top of MapReduce. It was created at Yahoo to solve the problems for developers who need to write MapReduce programs but do not have experiences coding in Java or MapReduce. Pig uses PigLatin; a dataflow scripting language. The Pig interpreter runs on the client machine and turns PigLatin script into a series of MapReduce jobs. Furthermore, Pig provides many features for analyzing data without Mapreduce code such as joining datasets, grouping data, position referring, loading non-limited data, etc.

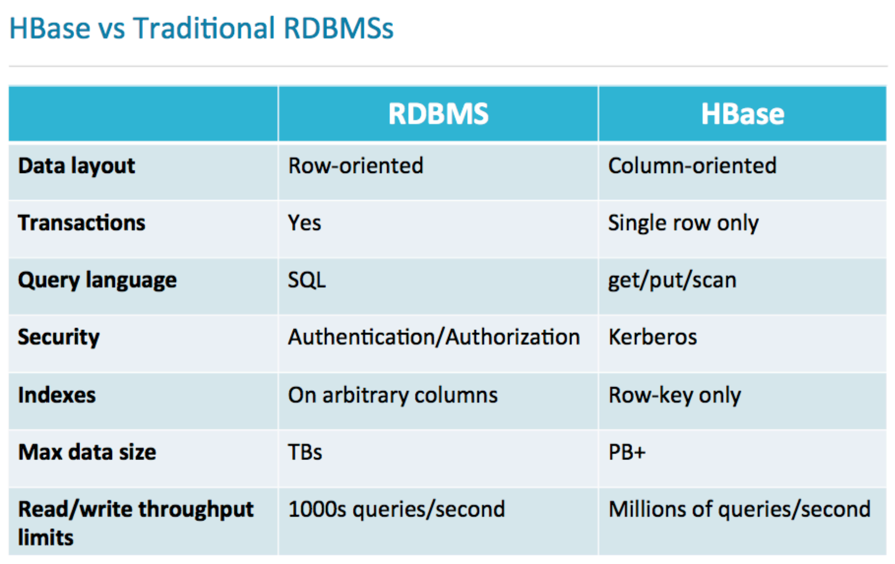
In Pig, a single data element is an atom. A collection of atoms is called a tuple such as a row or partial row. Bags are the collection of tuples. At the beginning, a PigLatin script will load datasets into bags, and then new bags are created using modification for those it already has. One can use the Grunt shell to run PigLatin in interactive or batch mode. Grunt shell also support DFS commands for Hadoop file system.

Pig and Hive has its own strengths and weaknesses. Users who have SQL background leaning toward Hive, whereas users with no SQL experience prefer Pig. Some organizations may choose both. Pig is good at manipulating less-structured data and turn into a more structured format. To query structured data, Hive is the preferred choice.

Flume (http://flume.apache.org) is a distributed service that provides an efficient method to gather data from possibly multiple sources (i.e. web servers, mail servers, firewalls) and inserting them into HDFS as they are generated. This means that there is no need to do batch-processing data later. Flume is an open source, which is firstly developed by Cloudera. The objectives for Flume's design include reliability, scalability, manageability and extensibility.

Sqoop (http://sqoop.apache.org) Sqoop is a short version for "SQL to Hadoop". It is a tool designed for transferring data between Hadoop and structured data such as a relational database. This means that tables from relational database can be imported into HDFS using a method offered by Sqoop. Furthermore, we can also populate database tables from files in HDFS.

HBase (http://hbase.apche.org) or Hadoop database. It is a distributed, non-relational database built on top of HDFS. It is inspired by Google's BigTable. It can store large quantities of data (i.e. petabytes) in a table. It offers very high write throughput scales for insertion. The tables can have thousands of columns even if there are sparse data. The index is go by column.



HBase vs Traditional RDBMSs

	RDBMS	HBase
Data layout	Row-oriented	Column-oriented
Transactions	Yes	Single row only
Query language	SQL	get/put/scan
Security	Authentication/Authorization	Kerberos
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	PB+
Read/write throughput limits	1000s queries/second	Millions of queries/second

Source: Cloudera

Oozie (http://oozie.apache.org) is a workflow engine and scheduler system to manage jobs on a Hadoop cluster. Workflows can be triggered by time (i.e. run every hour) or events (run when the input data exists). A workflow is a collection of actions such as Hadoop MapReduce jobs and Pig jobs. In addition, the workflow is arranged in a control dependency directed acyclic graph. The term control dependency can be described as one action depends on the other action. The second action can start the process only if the first action is completed.

Mahout (http://mahout.apache.org) is a library for machine learning and written in Java. The algorithms employed in Mahout supports data science in these areas: collaborative filtering, clustering, classification, and frequent item set mining. Some algorithms can be used as stand-alone programs, while many are optimized when work with Hadoop. The following table shows some algorithms in Mahout. The latest information about algorithms can be found in http://mahout.apache.org

Collaborative filtering	**Clustering**	**Classification**	**Frequent Item set**
Pearson correlation	K-means clustering	Bayesian	Parallel FP Growth
Log likelihood	Fuzzy K-means	Random Forest	
Spearman correlation	Canopy clustering	Support vector machine (SVM)	
Tanimoto coefficient	Latent Dirichlet Analysis	Naïve Bayes	
Singular value decomposition (SVD)		Stochastic gradient	

Mahout has some pre-built scripts for analyzing data. In addition, the libraries do not concern what items you want to recommend. It can be a recommender for friend, game, music, book, movie, etc.

The Hadoop ecosystem is fast evolving. Recently, focus has shifted from batch-oriented MapReduce to using in-memory technologies for real-time analytics. Impala and Spark are examples of these. Impala is similar to Hive but with the goal of providing faster results. Spark is a completely new approach to Big data that is particularly useful for iterative and acyclic data-flow problems. Especially, those that will benefit from the ability to retain or rebuild data quickly within the memory of a cluster node. For instance, at the time of writing this content, Mahout is no longer developed to utilize MapReduce, but will now focus on using Spark.

References

Cloudera training (n.d.).*The Hadoop Ecosystem: Learn about the projects comprising an entreprise data hub.* Retrieved from http://www.cloudera.com/content/cloudera/en/training/library/apache-hadoop-ecosystem.html.

Sammer, E. (2012). *Hadoop operations: A guide for developers and administrators.* O'Reilly Media. Retrieved from http://shop.oreilly.com/product/0636920025085.do.

The Hadoop Ecosystem Table. Retrieved from http://hadoopecosystemtable.github.io/.