Introduction to Big Data

David Lauzon
Departement of Software and IT Engineering
Ecole de Technologie Superieure (ETS), Universite du Quebec
Montreal, Quebec, Canada

December 24, 2012

Abstract

This paper presents an introduction to some of the most popular ”Big Data” technologies. ”Big Data” is a term coined to describe volume of data too high to be served by standard RDBMS or OLAP technologies, thus requiring alternative approaches. These approaches are usually regrouped as NoSQL technologies. This paper presents an overview of the Hadoop ecosystem, and some of its main components (HDFS, Hive, MapReduce, Impala, HBase). While HDFS provides the distributed filesystem; MapReduce provide the core framework to query HDFS data. Hive enables to query HDFS in batch using a subset of the SQL language. Impala does the same as Hive, but attemps to do it in real time. HBase is a distributed column-based storage, extremely efficient in retriving information based on a subset of a column key. We will then cover some uses cases when it is appropriate to use these technologies.

Keywords: Hadoop, Hive, HBase, Impala, BigData

1 Introduction

The term database means a structured collection of data. ”The data are typically organized to model relevant aspects of reality, in a way that supports processes requiring this information.” [41]. Databases are presents everywhere, from financial to medical systems, containing critical information to the function of our society. The size of the database varies greatly, and may go from few excel sheets (like finances of a small company) to petabytes like in a human genome. However a database simply contains data. In order for a database to be useful, there need to be a program(s) which enable us to store, modify, and extract information from the database. This set of programs is named a DBMS (database management system) [42].

The most popular DBMS type is RDBMS (Relational database management systems). RDBMS stores data in tables (e.g. a table employee that stores data for all employees), which provides a way to enforce links between the tables, and a language to query useful information from these tables.

As the amount of data increases, the time it takes to query data from RDBMS becomes very long, and the hardware cost required to process data becomes exponential. The only way to solve this problem is to change architecture. In the case that the query requires aggregates over one or more dimension(s) - as for example, the cost of apples for an apple juice factory in February 2010 - an OLAP approach may be used. However what do you do if you need to search through terabytes of log files to find for a particular error? Or, if you want to retrieve in real time all messages of a user from a database containing billions of messages? These kind of problems are the domain of BigData and NoSQL technologies.

2 RDBMS (Relational Database Management Systems

The relational model was invented in 1970 by Edward. F Codd at IBM Research Laboratory [3]. It represents data as tuples (e.g. list of elements, or rows), grouped together in tables. Each column represents a domain. An example of such tables is shown in figure 1.

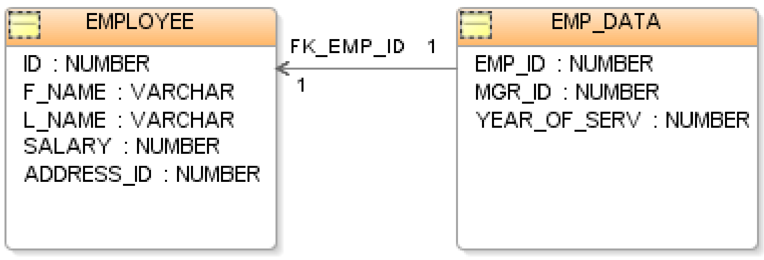


Figure 1: Example of two related tables. Source: [42]

In Codd's paper, the term "relation" is used to refer to a table. In this document, the term "relation" will be used to to describe relations between the tables. An RDBMS is based upon Codd's relational model and contains some important features. Each of these feature provides some benefits, but at same time induce an overhead cost. Understanding these costs and benefits is mandatory to decide if an RDBMS is the appropriate tool to solve a particular problem. The following benefits will be discussed : Transactions, ACID, Normalisation, SQL.

An important thing to mention is that RDBMS scale much better vertically [46], which basically means that as the amount of data increases you have to buy more and more expensive hardware.

2.1 Transaction

As stated in [43] a transaction "comprises a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions". The main purposes of transaction is to provide isolation between programs accessing the database concurrently and to provide a correct recovery in case there is total or a partial failure of a transaction. A transaction is ACID (2.2) by definition.

While transactions are extremely important in preserving the integrity of a database, they are expensive in terms of resources. As shown in (Harizopoulos and al. [19]), in terms of instructions required on a single node, transactions mechanism costs around 60 times more than without transactions.

2.2 ACID

ACID stands for Atomicity, Consistency, Isolation, Durability [37]. Here is a summary of its definition:

Atomicity requires that that transaction is "all or nothing", if it fails (or part of it), the database is left unchanged.[37]

Consistency "ensures that any transaction will bring the database from one valid state to another" [37].

Isolation "ensures that the concurrent execution of transactions results in a system state that could have been obtained if transactions are executed serially, e.g. one after the other" [37].

Durability "means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors" [37].

As described in the Transaction section, ACID properties comes with a strong overhead.

2.3 Normalisation

The idea of normalization is to remove data redundancy. This is usually made by removing the redundant data from one or more tables and move it to another table. Then create a link between the 2 tables [3]. During the query process, these tables are joined together. The advantages are reduced space, and quick data pruning in case the size of one of the table is limited. However joins are expensive operations, and as the size of tables grow to a certain point, the joins cannot be performed in memory anymore, and the speed of this operation decreases drastically. For a list of some joins techniques, please refer to (Oracle database performance tuning guide [25]).

There is different techniques to optimise join speed, like indexing and table partitioning, but they all have theirs costs and do not always apply.

2.4 SQL

SQL (Structured Query Language) is a "programming language designed for managing data in RDBMS" [47]. It enables creating table schemas, insert, updating, removing data in table. It also enable to query data from different tables, using a rich set of features. While it is a standard, different RDBMS vendors have different interpretations of it, thus there is minor variations in syntax across different products.

The advantages of SQL is that it is simple to use, and widely thought, thus a company using an RDBMS have very few training cost to pay, and can easily find developer to create / query using SQL. For more information about SQL please see (Wikipedia [47]).

3 OLAP Systems

OLAP (Online analytical processing) "is an approach to answering MDA(muti-dimensional analytical) queries swiftly." [45]. OLAP should be viewed more as an architecture to solve the problems of viewing aggregation of factual data depending on useful facets (dimensions). The data is stored in an OLAP cube, which consists of numeric facts, which are categorized by dimensions [45]. Figure 2 shows an example of an OLAP cube. The measures are (Units, Sales, Cost, Margin) while the dimensions are Products and Time. Each dimension can be further subdivided to a desired degree of granularity. In this example, Time is divided by month and Product is divided by Product Names.

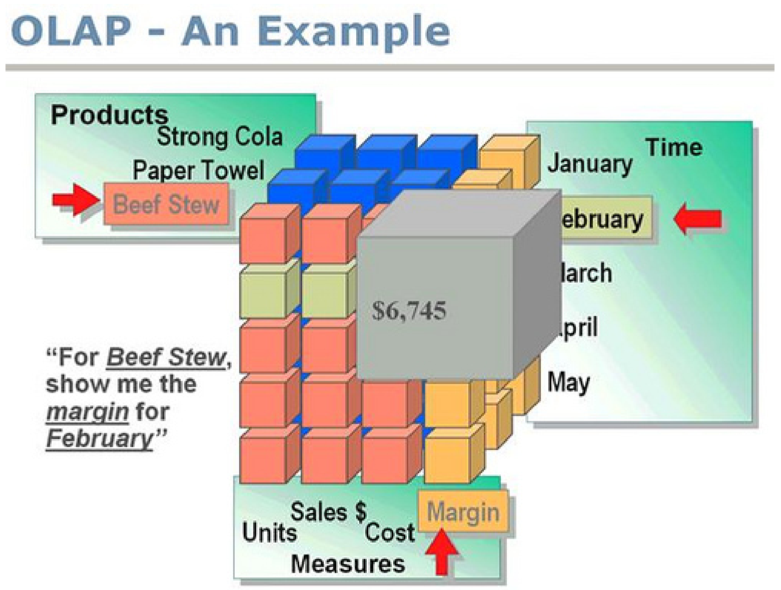


Figure 2: Example of an OLAP cube. Source: [18]

OLAP cubes are extremely efficient, and could be around 1000 times faster for producing complex queries than with equivalent OLTP relational data [45]. The performance gains come from the facts that the aggregation in many cells are pre-calculated.

The drawbacks is that 1) the complexity of queries is limited [27], and 2) the data needs to be formatted before being entered in the cube. To see the limitations of the first point it is simpler to analyze what

MDX (MultiDimensional eXpressions) language can and can not do. MDX has become a de facto standart [45]. The details on these limitations go beyong the scope of this document.

The data entry and formatting is usually performed using ETL (Extract, Transform, Load) procedures. Before building the cube, the data needed by the client needs to be processed using ETL prior to be available for queries. So the whole ETL cost has to be paid upfront. As an example let's say that on Figure 2 the cost data for beef stew on February is erroneous (for exemple: formatted in a different currency). What should we do ? Leaving it blank will cause errors in other aggregations (like total cost for all products in February); while removing the rows will affect margin, sales ... costs. There is no easy way to indicate that this data is erroneous and that result with it should not be used. Thus, this data will need to be cleaned. Even though it is possible that nobody will never use it, or that the impact is not important. All this clean up upfront represents a very high cost, and should be considered before before building an OLAP cube solution.

There is also a reason, the back end architecture behind an OLAP solution was not mentioned. Different technologies exists, some based on ROLAP (Relational OLAP), other on MOLAP(Multidimensionnal OLAP). Since there is a multitude of back-end and that particular back end does not solve both points 1) and 2) mentionned earlier, it was decided not to cover it in the present document.

4 Challenges with traditional storage

The advantages of RDBMS were presented in section 2. They work very well, but since they are vertically scaled, as the amount of data / users increases, the performance quickly degrades. In order to increase performance, either very expensive software and hardware have to be bought, or some of RDBMS advantages have to be dropped. As an example, tables start to get denormalized, some indexes are removed and some ACID properties may be dropped. All these optimisations, make the solution less flexible, harder to maintain and in fact doing what it was not designed to do.

While an OLAP solution, may help to work around these limitations they may be unsuitable or too costly to implement.

These are the kind of problems which could be solved by "Big Data".

5 So what is "Big Data" ?

"Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesnt fit the structures of your database architectures. To gain value from this data, you must choose an alternative way to process it." [8]. It is important to note that the appropriate Big Data solution for a use case is relative to the business value of each aspect. If your data doesn't fit the rigid structure of a DBMS, you may gear towards NoSQL (section 7) ; while if you have many PB that requires batch processing you may be more interested in Hadoop (section 8). Also this decision may be impacted by the cost of an advanced conventional database system.

6 Big Data Use Cases

To have a better understanding of what big data is, some big data problems and solutions will be presented in this section. This is an example of a company named Wireclub, which migrated from using Microsoft SQL server to MongoDB. The main advantages of switching to MongoDB were that it was free, faster then the previous solutions and had Freedom from rigid schemas (In other words you could store documents with any format), etc ... The drawbacks were the lack of tools, maturity, transactions and the lack of SQL like language [15]. But overall, the cost of scaling their RDBMS solution outweight the cost of coping with the drawbacks of their NoSQL solution. Many other examples could be listed, but they all come out with a similar conclusion.

7 NoSQL

NoSQL translates best as "Not Only SQL" : DBMS which generally do not use SQL for data manipulation and does not necessary contain structured data (as opposed to a DBMS table) [44]. Usually, this price is paid to gain better performance, better scaling or both. Since performance and scalability arise with BigData, NoSQL databases become a BigData solution. Since there is a large variety of NoSQL software, we will group them by the solutions they offer.

7.1 Document Store

Documents stores are centralized in storing "documents". A document encapsulate and encode data in various formats (depending on vendors). Documents have a unique id, but a document store provides a language (or API) enabling the user to query by document ID or contents of the document. Some examples of document stores are Couchbase Server and MongoDB.[44]. As an example storing images in a document, with the creator name, and some other info like, type of image, a description and some other user created meta data would be a good use case for a document store.

7.2 Key-Value store

The idea of a key-value store is to separate the data into two parts: data and a key to access the data. The data format is irrelevant. The idea is to access the data using the key, or a prefix subset of a key. There is plenty of different implementations, for example some holds all their data in RAM like MemcacheDB, while others can be persistent on disk like HBase (see 8.2.6) [44]. Key value efficiency depends on finding a good key to retrieve a value. As an example, if you store user informations, using user ID as a key is a good use case for a key value store.

Hadoop and the systems based on it are considered NoSQL, and will be described in the following sections.

It is important to remember that there is other solutions, and a more complete research should be made in case the present solutions does not meet a particular use case.

The important part to remember is that tradeoffs must be weighted before choosing any solution.

8 Hadoop

Many newcommers to Hadoop are intimidated by its complexity and wonder where to get started. This section attempts to introduce the Hadoop ecosystem in a progressive way that is easy to learn. The first part describes briefly the core parts of Hadoop, and what are the most popular components that can be added to it. Then, five of these components will be explained in more details. They are: HDFS (8.2.1), Map-Reduce (8.2.2), Hive (8.2.3), Impala (8.2.4), and HBase (8.2.6).

8.1 Overview of the Hadoop ecosystem

Hadoop is the system you want to use to scale horizontally. Hadoop is batch processing system that allows to search through files in a distributed manner. By distributing files automatically accross the cluster, Hadoop can crunch massive amounts of data in a short time. Also, since it uses the "write-once" paradigm, Hadoop's write throughput is very high. Also, Hadoop performs many times faster if you can store fewer large files as opposed to many small files.

8.1.1 History

The Hadoop project was started around 2006 by Doug Cutting, a Yahoo architect, who was interested in scaling a distributed search engine crawler called Nutch. When Google released the GFS [17] and Map

Reduce [6] papers, Doug saw a potential and began implementing some of Google's ideas into Nutch. Eventually, Hadoop spawned into its own project and it's development is now being lead by the Apache Software Foundation. The unusual name Hadoop comes the toy elephant of Doug's son.

Facebook quickly started using Hadoop, and contributed many other projects to the Hadoop ecosystem[1]. Others Hadoop users include Amazon.com, LinkedIn, Twitter, StumbleUpon, Apple, HP, IBM, Google, Netflix, Microsoft and even the US National Security Agency! [38][20]

8.1.2 Some quick facts about Hadoop

- On november 8th 2012, Facebook announced that "Over half a petabyte of new data arrives in their [Hadoop] warehouse every 24 hours"[9].
- As of december 18th 2012 according to Indeed.com, the average salary of a Hadoop Engineer in San Mateo, CA is of $134,000 per year[21].

8.1.3 Distributions

Since Hadoop is open source software, you can download it directly from Apache Software Foundation. But there is also what the so called "distributions" for Hadoop, which aims to bundle the different Hadoop components into package for easier integration. The most popular distributions are:

1. Cloudera Distribution Including Hadoop (CDH), which can be downloaded free of charge. Cloudera sells technical support and Hadoop management tools. CDH has been distributed since march 2009 [32] and has hired many Hadoop core committers.
2. Hortonworks Data Platform (HDP), which is also open source. HortonWorks has partnered with Microsoft to make Hadoop compatible on Windows Azure [12].
3. MapR Distribution. MapR Technologies rewrote Hadoop's core (HDFS and MapReduce) claiming to achieve significantly higher performance and security, while still providing the standard eco-system components [33].

8.2 Hadoop Components

8.2.1 HDFS

HDFS (or Hadoop Distributed File System) was modeled on the Google File System paper. Google describes GFS as "a scalable distributed file system for large distributed data-intensive applications" which aims to achieve two main goals [17] :

1. Provide fault tolerance while running on inexpensive commodity hardware.
2. Delivers high aggregate performance to a large number of clients.

HDFS has been known to scale up to a few thousands nodes, but in some cases it may makes some sense to divide a cluster into two or more, for exemple to serve different regions of the world. In this case, replications (of some data) across data centers may be required.

[1]It is generally accepted to use the name "Hadoop" for the two core components which are HDFS (the file system) and Map Reduce; and to use the name "Hadoop ecosystem" to subsume Hadoop and all the components built on top of Hadoop.

8.2.1.1 HDFS Architecture

The documentation explains as follows: "HDFS has a master/slave architecture. An HDFS cluster consists of a single **NameNode**, a master server that manages the file system namespace and regulates access to files by clients. In addition, there is many **DataNodes**, usually one per node in the cluster, which manage storage attached to the nodes that they run on.

HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file systems clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

[...]

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The **block size** and **replication factor** are configurable per file, specified at file creation time or can be changed later" [13].

HDFS is rack-aware, meaning that it will tries to replicate a block on multiple racks, so that if rack becomes temporarily unavailable (e.g. a switch failure), the data can still be accessed from another rack. Futhermore, in order to minimize the network traffic, the client send only one copy of the block to an HDFS datanode; it is the datanode itself which replicates the block to the other datanodes.

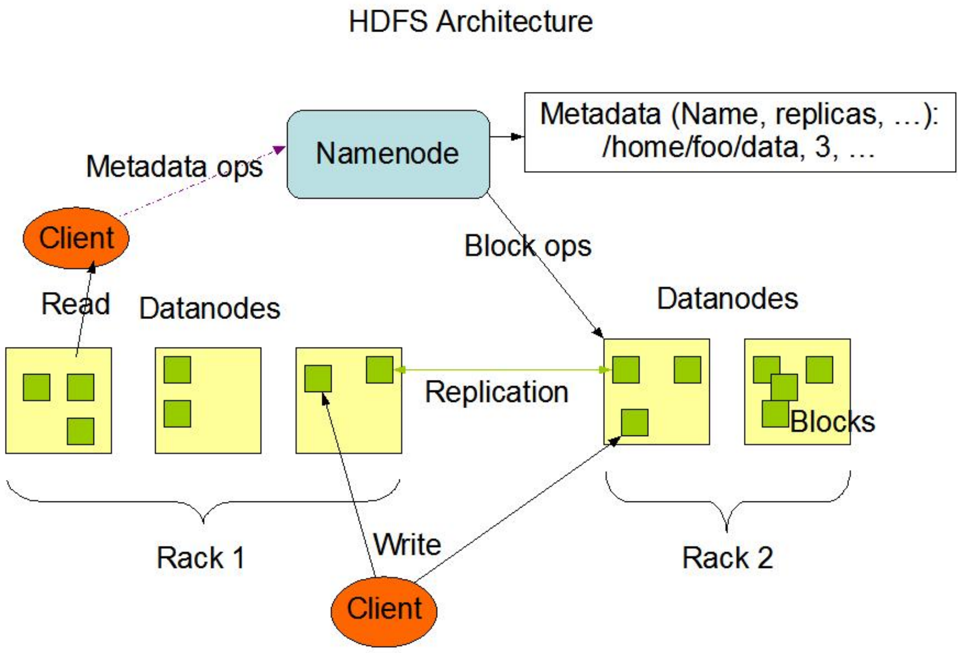


Figure 3: HDFS Architecture. Source : [13]

These operations can be visualized on Figure 3. Some other interesting facts to note about HDFS:

- HDFS performs up to 30% faster **without** RAID [2]. However, typically RAID is used on disks where the NameNode metadata is stored to protect against corruption.
- Default replication factor is 3, so by default there is 3 copies of each file in the cluster.

[2]Tom White: "In RAID-0 read and write operations are limited by the speed of the slowest disk in the RAID. While in [HDFS replication] disk operations are independent, so the average speed of operations is greater than that of the slowest disk. [...] Furthermore, [unlike RAID] HDFS can continue to operate without a failed disk." [36, p. 260].

- Default block size is 64 MB. This setting helps providing a higher throughput as it reduces the number of disk seeks.
- HDFS is write-once, read-many times. However append operations are supported.
- HDFS does not provide a caching layer. Since HDFS sits on top of the operating system's file system, the cache can from the Linux kernel can kick in (where applicable).
- Higher read througput can be achieved by either increasing replication factor, adding disks to a node, or adding more nodes.

8.2.1.2 HDFS High Availability (advanced topic)

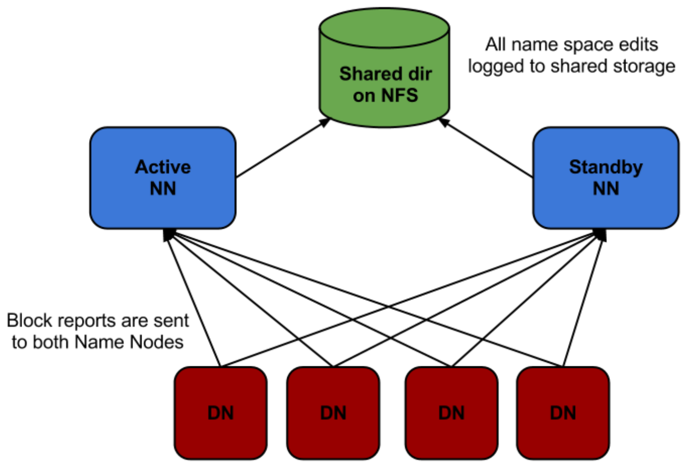
Prior to February 2012, Hadoop had one single point of failure (SPOF) : if the **namenode** has crashed, your entire cluster becomes unavailable until the namenode machine or an image is brought back online. This has now been fixed and there is two types of configuration available.

HDFS NameNode HA using NFS: First in February 13th 2012, CDH4 beta 1 allowed a configuration using NFS (as show on Figure 4a). In this setup, there is two namenode machines which share a common directory on NFS: one called the **Active Namenode**, and the other the **Standby Namenode**. "All mutations to the file system namespace, such as file renames, permission changes, file creations, block allocations, etc, are written to a persistent write-ahead log [on the NFS shared directory] by the [**Active**] **Name Node** before returning success to a client call.

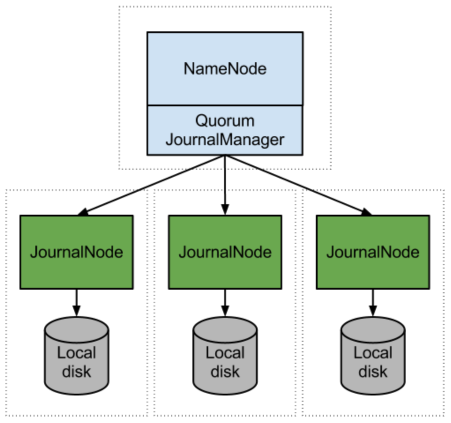
[...]

The **standby Name Node** polls the shared edits directory frequently, looking for new edits written by the **active Name Node**, and reads these edits into its own in-memory view of the file system state [(e.g. the namenode metadata)]" [24].

However the **namenode** does not store block locations, but is rather informed by the periodic block reports sent by the datanodes. And since "starting a Name Node from cold state can take [...] up to an hour to receive the necessary block reports from all Data Nodes in a large cluster" [24], all datanodes are configured to send block reports to both namenodes. This setup provides a "hot standby Name Node that can take over serving the role of the active Name Node with no downtime" [24].


Shared dir on NFS
All name space edits logged to shared storage
Active NN
Standby NN
Block reports are sent to both Name Nodes
DN
DN
DN
DN

(a) HDFS NameNode HA using NFS. Source: [24]



(b) HDFS NameNode HA using QuoromJournalManager. Source: [22]

HDFS NameNode HA using QuorumJournalManager: The concern with **HDFS HA using NFS** is that relying on shared storage requires custom hardware (e.g. NAS) that can be expensive, and involves complex deployment and monitoring. To remove these limitations, Cloudera designed a system called QuoromJournalManager where "the NameNode acts as a client and writes edits to a set of JournalNodes, and considers the edits committed when they have been replicated successfully to a majority of these nodes" [22]. Also a new component, called ZooKeeper Failover Controller (ZKFC), is installed on each potential namenode to monitor the health of the namenode. When a failure of the active namenode is detected, the ZKFC quickly triggers an election for a new active namenode.

The communication and locking mechanism of this setup is managed by ZooKeeper, another Hadoop component (described at section 8.2.5). The datanodes send block reports to all the (active and failover) namenodes.

8.2.2 Map Reduce

HDFS MapReduce was modeled on the *Google MapReduce: Simplified Data Processing on Large Clusters* paper which Google summarizes as :

> "MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.
>
> [...]
>
> Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the programs execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system" [6].

8.2.2.1 Map Reduce Architecture

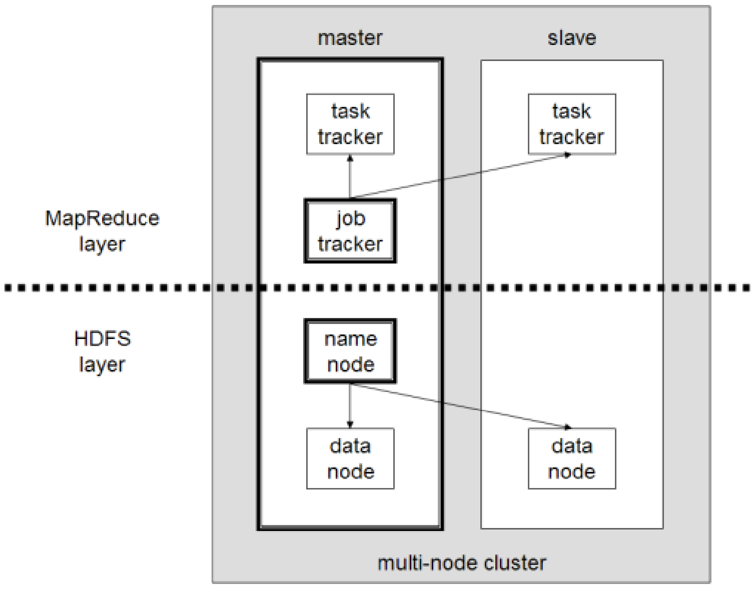


Figure 4: Simple overview of Hadoop core components

Hadoop's implementation of the Map Reduce paradigm functions exactly the same way as described above. As shown by Figure 4, two new daemons are introduce: "The **JobTracker** is the central coordinator of jobs in MapReduce. It controls which jobs are being run, which resources they are assigned, etc. On each node in the cluster there is a **TaskTracker** that is responsible for running the map or reduce tasks assigned to it by the **JobTracker**" [16].

To expand on this definition, here's the five phases of a map-reduce job which I summarize from the *Appendix B. Overview of Hadoop* of the *Programming Pig* book by O'Reilly [16]:

1. "In the **map phase**, MapReduce gives the user an opportunity to operate on every record in the data set individually. This phase is commonly used to project out unwanted fields, transform fields, or to apply filters. [...] Every MapReduce job specifies an InputFormat. This class is responsible for determining how data is split across map tasks and for providing a RecordReader. [...] Each InputSplit is given to an individual map. [As] the InputSplit includes a list of nodes [which holds the HDFS data to be read], MapReduce is able to move the computation to the data [to the node which contains the data and minimize transfert of data between the nodes]".

2. "The **combiner** gives applications a chance to apply their reducer logic early" [and to reduce the size of the input data when the memory buffer fills up and is about to spill to disk].

3. "During the **shuffle phase** MapReduce partitions data among the various reducers".

4. "The input to the **reduce phase** is each key from the shuffle plus all of the records associated with that key. Since all records with the same value for the key are now collected together, it is possible to do joins and aggregation operations such as counting".

5. "The **output phase** includes serializing, possibly compressing, and writing [the records] to HDFS, HBase, etc" .

8.2.2.2 Map Reduce Example

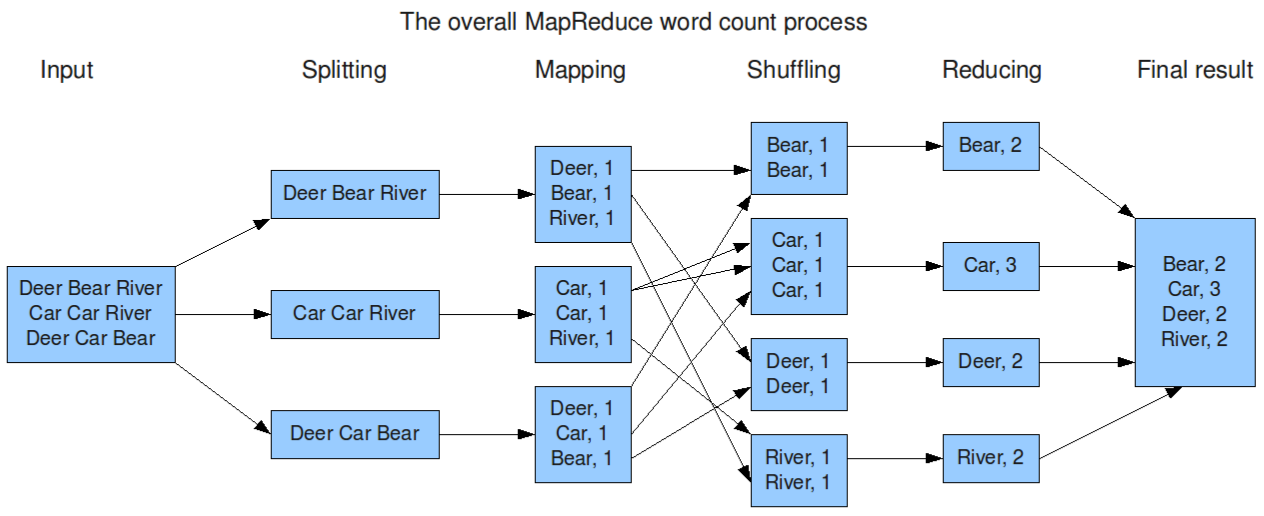


Figure 5: Example of a Map Reduce Job. Source: [35]

You may find easier to understand this process with the classic word count example on Figure 5. Let's imagine the input is a CSV file having its fields separated by a space, and its rows by a carriage return.

8.2.3 Hive

Hive allows users to query HDFS files using an SQL-like language. Hive achieve this by creating a metastore database to hold the tables' definition : the list of its columns along with their datatypes, as well as the mapping between the table and the folder in HDFS.

Hive converts turns the SQL query into map reduce jobs, executes them, and return the result to the client, sometimes outputting back to HDFS. Several types of join are supported and subqueries as well. Hive provides many builtin functions, as well as allowing to write you own UDF (User Defined Functions).

Hive supports many file formats from CSV file to your own custom file format and can read from HBase tables. Many compression codecs can be used with Hive, notably LZO, Snappy, GZip, and BZip. When the objective is to minimize the I/O bandwith, LZO or Snappy block compression are likely to offers the best **decompression speed** / **compressed size** ratio.

Dividing your table in several partitions is a great way to minimize the size of data scanned by the queries. For exemple, if most of your queries uses a date field and/or product sku, you may gain performance in partionning your table on these columns. However, beware that over partitionning may yield into too many files and degrading HDFS performance.

Hive was originely created by Facebook Data Infrastructure Team in 2008 [34].

8.2.4 Impala

Impala is similar to Hive in many ways:

- Usage of SQL to query HDFS files.
- Tables created with Hive can be queried with Impala.
- Both can query HBase tables.
- The ODBC driver is compatible.
- Hue (a web GUI for Hive and Hadoop) is compatible.

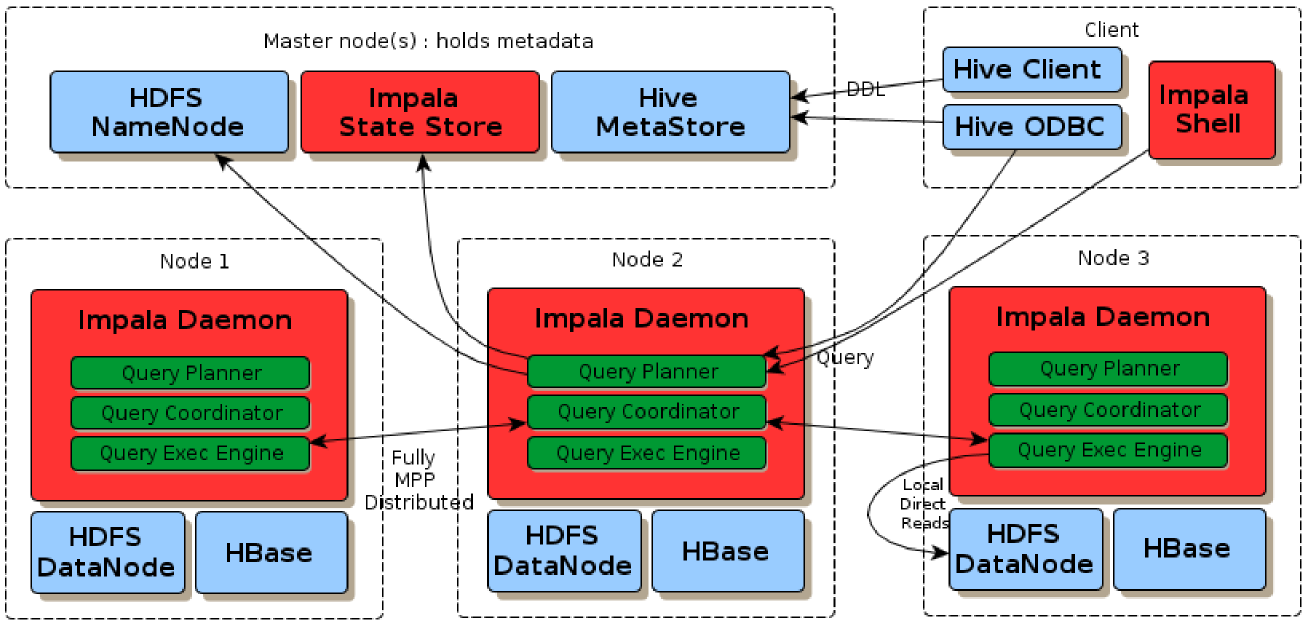


Figure 6: Impala Architecture

However the architecture is completely different. First of all, Impala is built in C++ allowing many memory optimisations that cannot be done in Java. Also Impala does not uses the map reduce framework. Instead it runs a daemon on each DataNode. By using the impala-shell, you can connect to any impala daemon and perform a SQL query. The daemon will distribute the work to the other daemons that holds the data to perform massively parallel processing (MPP).

By configuring your cluster HDFS to allow Impala to bypass the HDFS datanode, Impala can achieve incredible speed in ad hocs queries. Impala's strenght are: 1) Data locality and 2) no JVM startup cost or map-reduce overhead (e.g. dumping map output to disk, then read again by the reducers).

As of writing this paper, Impala is currently in beta version 0.3 and 1.0 GA version is planned for end of Q1 2013 [10].

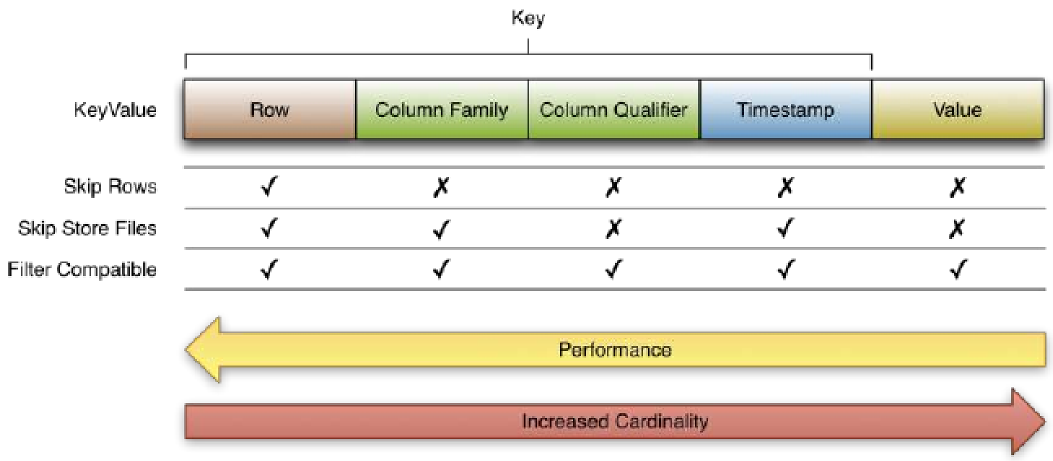
8.2.5 ZooKeeper

According to Apache Software Foundation: "ZooKeeper is a [highly available, fault tolerant, and persistent] centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications." [14].

In other terms, ZooKeeper is a highly available, fault tolerant, persistent, hierarchical distributed key-value store for data of small size. Some use cases of ZooKeeper includes HBase and Hadoop Name Node High Availability (section 8.2.1.2).

8.2.6 HBase

In this section we will concentrate on HBase data architecture. Since almost everything in this part is summarized in the excellent book "Hbase in Action" [7], we will skip references to it. The important part is to understand how it stores data. This is illustrated in Figure 7.

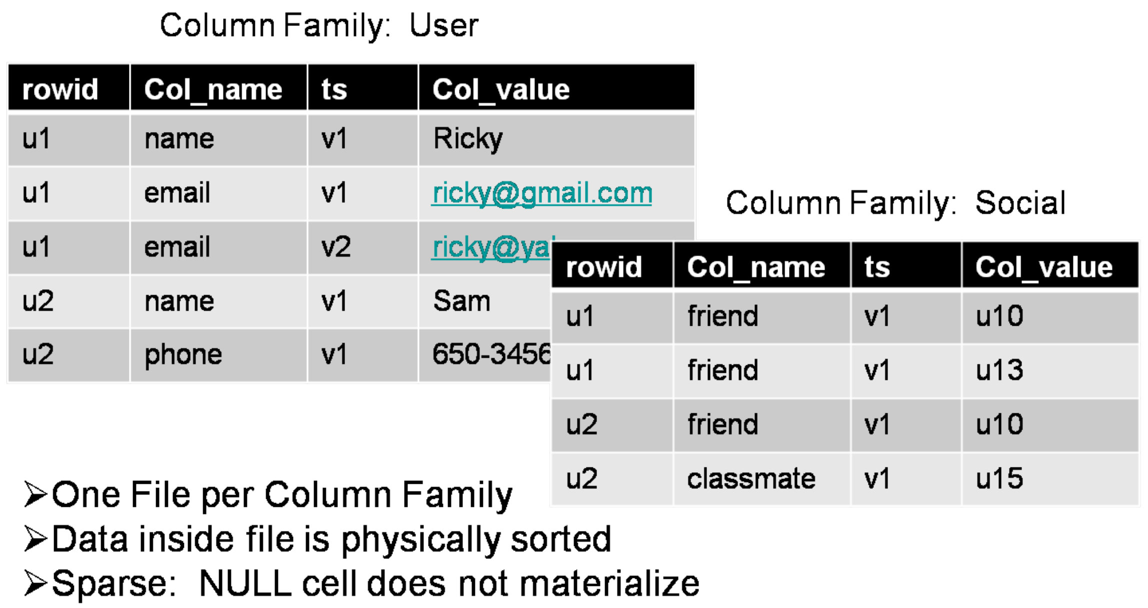


	Row	Column Family	Column Qualifier	Timestamp	Value
Skip Rows	✓	✗	✗	✗	✗
Skip Store Files	✓	✓	✗	✓	✗
Filter Compatible	✓	✓	✓	✓	✓

Figure 7: Key cardinality

The part **key** is used to access the data, while the part **value** stores the actual value associated with the key. **Row** is a unique row identifier created by the user. **Column Qualifier** is equivalent to a column in a table. The main difference between a RDBMS table and a column oriented table like in HBase, is that a columnar storage can 1) does not need to read the data for the columns that are not included in the search, and 2) empty (null) values takes no space. **Column family** is used to group columns together (we will discuss this point later). Finally timestamp is used to timely identify a value entry. HBase is designed as a versioning update system, i.e. every time a value is modified, it creates a new version with a timestamps.

The RDBMS tables equivalent would be as the following : **Row** = **Primary Key**, **Column Qualifier** = **Column**, **Value** = **Value** in a RDBMS table, and for each value there is an added timestamp. Figure 8 shows an example of HBase table.



Column Family: User

rowid	Col_name	ts	Col_value
u1	name	v1	Ricky
u1	email	v1	ricky@gmail.com
u1	email	v2	ricky@ya
u2	name	v1	Sam
u2	phone	v1	650-3456

Column Family: Social

rowid	Col_name	ts	Col_value
u1	friend	v1	u10
u1	friend	v1	u13
u2	friend	v1	u10
u2	classmate	v1	u15

- One File per Column Family
- Data inside file is physically sorted
- Sparse: NULL cell does not materialize

Figure 8: HBase table example.
Source: `http://horicky.blogspot.ca/2010/10/bigtable-model-with-cassandra-and-hbase.html`

The similarity ends here. HBase tables can only be efficiently queried by a key or a prefix of a key. If we take Figure 8 and we try to search for the following key : ”u1:User:name:v1” (rowid:column family,column qualifier:timestamp) we will get ”Ricky” as results. If we search for ”u1:User:email” we will get 2 values ”ricky@yahoo.com” and ”ricky@gmail.com”. However if we try to search for let’s say ”User:email:v2”, it will have to scan the whole table, before returning ”ricky@yahoo.com”. Also, there is also no automatic way to make joins in HBase (read lots of headaches to do manually, by using mapreduce jobs).

Thus the question about using HBase is about access patterns. If the client can build a key and the solution will search by the key or its prefix, then HBase should be used. Otherwise other BigData solutions are more appropriate. As an example, storing users and their tweets in an HBase tables would be a good solution for HBase, since virtually all queries are about a user having his user id and search his tweets, or other users and their tweets (So the rowid is basically composed of ”userid+timestamp”). A bad example would be to store YouTube videos, using the video title as rowid. This is bad since virtually all you tube videos are accessed by part of a title, and thus will require a full table scan from HBase.

More advanced topics about HBase talk about how to make your key (like for example hashing the key) so that the data is distributed evenly in the cluster, or how to group data in column families to increases access speed for commonly accessed data. Also, since row operations in HBase are atomic, transactions may be ”patched” into HBase, with some limitations. However these features, are only needed in case the Key creation described earlier applies to the business case.

9 Conclusion

In this paper we have presented some conventional database technologies such as RDBMS and OLAP, and described their strengths. The RDBMS are mostly with transactions, normalization and SQL, while for OLAP is quick queries over multidimensional data. While these technologies are mature, widely spread and intuitive to use, it comes a point when either the amount of data or the rate of data coming is too high or the structure of data is such, that its processing can not be done with the performance required by the client. Another important point is to evaluate is the cost of fixing these problems using either RDBMS or OLAP technologies. Then, the only choice left is to use Big Data technologies. Big data technologies are mostly NoSQL databases, which focus on solving a particular problem. We have seen for example that: MangoDB is very good at storing documents, Hadoop with Map-reduce can process huge amount of data in batch fashion, HBase is very efficient in working with data which is accessed by a unique key, while Hive and Impala offer an SQL-like language to work with tables in a Hadoop environement. All these solutions have their advantages and disadvantages, which should be weighted before choosing one. There is no "silver-bullet" in big Data. To finish it would be interesting to compare vendor specific Big Data solutions, like Windows Azure or MapR, and compare them with the other technologies explored in this research.

Acknowledgment

I thank my teacher Alain April for giving me the opportunity to personally experiment with big data technologies like Hadoop, Impala, Hive and HBase. As well, I thank my friend Anton Zakharov for reading and commenting my research.

References

[1] Spyros Blanas, Yinan Li, and Jignesh M. Patel. Design and evaluation of main memory hash join algorithms for multi-core cpus. 2011.

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, and Google. Bigtable: A distributed storage system for structured data. 2006.

[3] Edgar Frank Codd. A relational model of data for large shared data banks. 1970.

[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford, and Google. Spanner: Googles globally-distributed database. 2012.

[5] OLAP Council. Olap council white paper. 1997.

[6] Jeffrey Dean, Sanjay Ghemawat, and Google. Mapreduce: Simplified data processing on large clusters. 2004.

[7] Nick Dimiduk. *HBase in action*. Manning, Shelter Island, NY, 2012.

[8] Edd Dumbill. Making sense of big data. 2012.

[9] Facebook Engineering. Under the hood: Scheduling mapreduce jobs more efficiently with corona. `https://www.facebook.com/notes/facebook-engineering/under-the-hood-scheduling-mapreduce-jobs-more-efficiently-with-corona/10151142560538920`, Nov 2012. Retrieved Nov 28th, 2012.

[10] Justin Erickson. What's next for cloudera impala? `http://blog.cloudera.com/blog/2012/12/whats-next-for-cloudera-impala/`, dec 2012. Retrieved Dec 18th, 2012.

[11] Andrew Fikes and Google. Storage architecture and challenges. 2010.

[12] Forbes. Microsoft launches hadoop for windows server and azure. http://www.forbes.com/sites/tomgroenfeldt/2012/10/24/microsoft-launches-hadoop-for-windows-server-and-azure/, oct 2012. Retrieved Nov 28th, 2012.

[13] Apache Software Foundation. Hdfs architecture. http://hadoop.apache.org/docs/r0.20.2/hdfs_design.html. Retrieved Dec 14th, 2012.

[14] Apache Software Foundation. Welcome to apache zookeeper. http://zookeeper.apache.org/. Retrieved Dec 18th, 2012.

[15] Rod Furlan. Migrating from microsoft sql server to mangodb - lessons learned.

[16] Alan F Gates. Appendix b. overview of hadoop. http://ofps.oreilly.com/titles/9781449302641/hadoop_overview.html, sep 2011. Retrieved Dec 14th, 2012.

[17] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, and Google. Gfs: The google file system. 2003.

[18] Issam El Hachimi and Ludovic Schmieder. La business intelligence. http://www-igm.univ-mlv.fr/~dr/XPOSE2010/Extract_Transform_Load/bi.php. Retrieved Nov. 5th, 2012.

[19] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. Oltp through the looking glass, and what we found there. 2008.

[20] J. Nicholas Hoover. Informationweek: Nsa submits open source, secure database to apache. http://www.informationweek.com/government/enterprise-applications/nsa-submits-open-source-secure-database/231600835, sep 2011. Retrieved Nov 28th, 2012.

[21] Indeed.com. Hadoop engineer salary in san mateo, ca. http://www.indeed.com/salary/q-Hadoop-Engineer-l-San-Mateo,-CA.html, Dec 2012. Retrieved Dec 18th, 2012.

[22] Todd Lipcon. Quorum-based journaling in cdh4.1. http://blog.cloudera.com/blog/2012/10/quorum-based-journaling-in-cdh4-1/, oct 2012. Retrieved Dec 14th, 2012.

[23] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, and Google. Dremel: Interactive analysis of web-scale datasets. 2010.

[24] Aaron Myers. Cloudera blog: High availability for the hadoop distributed file system (hdfs). http://blog.cloudera.com/blog/2012/03/high-availability-for-the-hadoop-distributed-file-system-hdfs/, mar 2012. Retrieved Dec 14th, 2012.

[25] Oracle. Oracle database performance tuning guide: Hash joins. http://docs.oracle.com/cd/B28359_01/server.111/b28274/optimops.htm#i76073. Retrieved Nov. 5th, 2012.

[26] Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan, and Google. Interpreting the data: Parallel analysis with sawzall. 2005.

[27] Philip Russom. Next generation data warehouse platforms. 2009.

[28] Philip Russom. Analytic databases for big data. 2012.

[29] Luca Santillo. Size & estimation of data warehouse systems. 2001.

[30] Michel Schneider. Well-formed data warehouse structures. 2003.

[31] Dr. Pushpa Suri and Meenakshi Sharma. A comparative study between the performance of relational & object oriented database in data warehousing. 2011.

[32] TechCrunch. Cloudera raises $5 million series a round for hadoop commercialization. `http://techcrunch.com/2009/03/16/cloudera-raises-5-million-series-a-round-for-hadoop-commercialization/`, mar 2009. Retrieved Nov 28th, 2012.

[33] MapR Technlogies. The open, enterprise-grade distribution for apache hadoop. `http://www.mapr.com/products`. Retrieved Nov 28th, 2012.

[34] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Facebook data infrastructure team. 2008.

[35] Martijn van Groningen. Appendix b. overview of hadoop. `http://blog.jteam.nl/2009/08/04/introduction-to-hadoop/`, aug 2009. Retrieved Dec 14th, 2012.

[36] Tom White. *Hadoop : the definitive guide (2nd Edition)*. O'Reilly, Farnham, 2010.

[37] Wikipedia. Acid. `http://en.wikipedia.org/wiki/ACID`, nov 2012. Retrieved Nov. 5th, 2012.

[38] Wikipedia. Apache hadoop. `http://en.wikipedia.org/wiki/Apache_Hadoop`, 2012. Retrieved Sep 17th, 2012.

[39] Wikipedia. Big data. `http://en.wikipedia.org/wiki/Big_data#Definition`, nov 2012. Retrieved Nov. 5th, 2012.

[40] Wikipedia. Big data. `http://en.wikipedia.org/wiki/Big_data#Definition`, nov 2012. Retrieved Nov. 5th, 2012.

[41] Wikipedia. Database. `http://en.wikipedia.org/wiki/Database`, nov 2012. Retrieved Nov. 5th, 2012.

[42] Wikipedia. Database management system. `http://en.wikipedia.org/wiki/Database_management_system`, nov 2012. Retrieved Nov. 5th, 2012.

[43] Wikipedia. Database transaction. `http://en.wikipedia.org/wiki/Database_transaction`, nov 2012. Retrieved Nov. 5th, 2012.

[44] Wikipedia. Nosql. `http://en.wikipedia.org/wiki/NoSQL`, nov 2012. Retrieved Nov. 4th, 2012.

[45] Wikipedia. Olap. `http://en.wikipedia.org/wiki/OLAP`, sep 2012. Retrieved Sep. 23rd, 2012.

[46] Wikipedia. Scalability. `http://en.wikipedia.org/wiki/Scalability#Scale_vertically_.28scale_up.29`, nov 2012. Retrieved Nov. 5th, 2012.

[47] Wikipedia. Sql. `http://en.wikipedia.org/wiki/SQL`, nov 2012. Retrieved Nov. 5th, 2012.