Université Libre de Bruxelles

INFO-H-419: Data Warehouses

Hadoop in Data Warehousing

Author:
Alexey Grigorev

January 23, 2014

Introduction

Today the amounts of data stored in Data Warehouses are becoming more and more enormous. While traditional ways of Data Warehousing design on top of Relational Databases are still popular, they fail to curb terabytes of data efficiently, which is mostly attributed to complexity of scaling relational databases. There are new emerging approaches that try to address this problem.

One of such approaches is to use the MapReduce paradigm and Hadoop as the implementation for building large Data Warehouses over distributed network of servers that can handle huge volumes of data. Hadoop has already become a proven tool for BigData analytics and now there is a rising interest in this technology for Data Warehousing purposes.

The goal of the work is to discuss in what ways Hadoop, as a MapReduce framework, can be used in Data Warehouses and then compare it with traditional approaches and see in which situations it should be beneficial to use Hadoop in a Data Warehousing project. Additionally we plan to see in what cases the traditional approaches should still be preferred over Hadoop.

This work is organized in the following way: firstly we introduce the MapReduce paradigm and Hadoop as the implementation. In the second section we discuss how to build a data warehousing solution entirely on top of Hadoop. And in the third section we analyze in what ways Hadoop and Data Warehouses can co-exist and how Hadoop can be incorporated into a Data Warehouse.

1 Hadoop

1.1 MapReduce

MapReduce is a paradigm of parallel computation that initially comes from Functional Programming. It is mainly used as a framework for large scale parallel data processing and typically runs on a distributed file system [10]. This framework is expressive enough to hide details of parallel execution and allow users to focus entirely on processing data.

The main primitives of this framework are two functions: **map** and **reduce**, and both of them must be provided by a programmer. A map function takes a key-value pair (`in_key`, `in_value`) and produces multiple key-value pairs called "intermediate result". The output of the map function is then grouped by keys and fed to the reduce function, which typically aggregates its input and produces the final result [1].

A *MapReduce job* is an execution of these two function on some data set. A job consists of three stages:

1. **the map stage**, where the map function is applied to each key-value pair of the input dataset
2. the grouping stages, at which the intermediate results are shuffled and combined, and
3. finally **the reduce stage**, where the reduce function is applied to each group of the intermediate result and produces the final result.

A job is executed in parallel with many nodes reading from many input sources (see Figure 1).

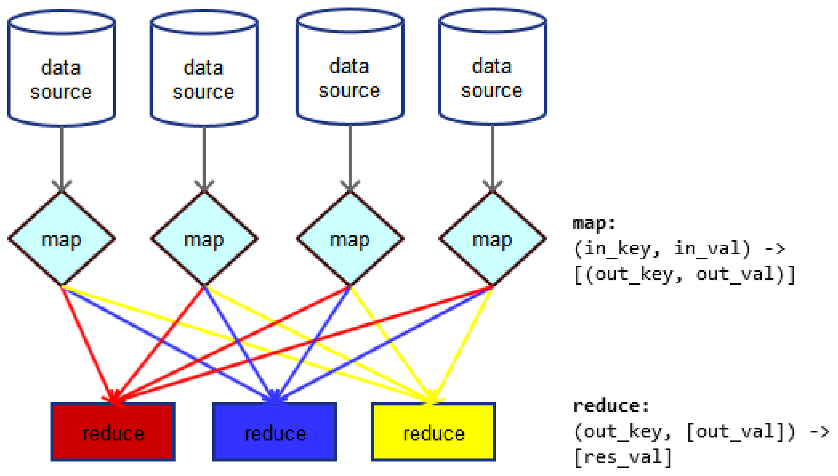

Figure 1: A MapReduce job: (1) map phase, (2) grouping, (3) reduce phase

Let us consider a classical example: given some text, we need to calculate how many occurrences of each word are there.

In a python-like pseudo-code the map and reduce functions are defined as follows

```
def map(String input_key, String doc):
  for each word w in doc:
    EmitIntermediate(w, 1)

def reduce(String output_key, Iterator output_vals):
  int res = 0
  for each v in output_vals:
    res += v
  Emit(res)
```

At the map phase `EmitIntermediate(`w`, 1)` outputs an intermediate result with word w and a associates a counter 1 with each word. At the reduce phase we receive the intermediate results grouped by keys. In this case it has the form $(w, [1, 1, ..., 1])$: a word w followed by a list of ones. We calculate how many ones there are in the result, this way obtaining the total count per word w in the text.

1.2 Hadoop as MapReduce Implementation

Hadoop is an open-source implementation of the MapReduce paradigm that is nowadays widely adopted . They position themselves as "... a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures" [12].

However the term "Hadoop" usually refers to the entire family of products, and the most important products are MapReduce and Hadoop Distributed File System (HDFS). Also the term sometimes may refer to other Hadoop-related products such as HBase [13], Hive [5], Mahout [14] and many others.

1.3 Hadoop MapReduce Component

Hadoop MapReduce Component is a data processing tool that works on top of HDFS and implements MapReduce. This essentially is the main component of Hadoop.

A Hadoop cluster consists of the main node, called *Name Node* that orchestrates the process of assigning jobs and monitoring the progress. A MapReduce job is done by data nodes, also called *workers*. There are two kind of workers: *mapper* workers and *reduce* workers, they work at the map and reduce phases respectively. The number of reduce workers R is usually know and specified beforehand.

A MapReduce job in Hadoop is executed in the following way (see Figure 2):

- The Name Node chooses some idle workers and assigns them a map task
- Before starting the map task, the workers need to do some preparation work:
  - The specified input file is loaded to the file system and partitioned into blocks (typically 64 kb)

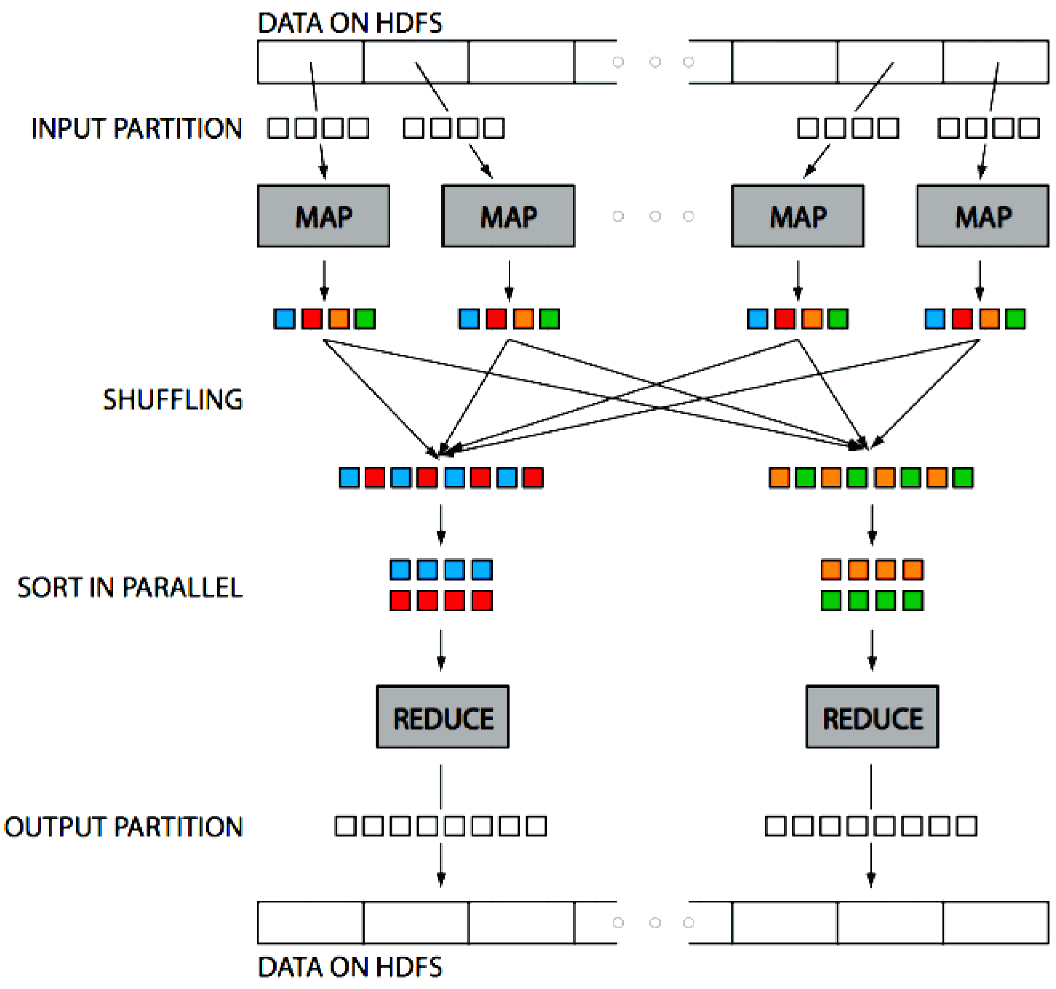

Figure 2: A MapReduce framework on Hadoop (figure source: [11])

  - Then each block is replicated three times to guarantee fault-tolerance

- The **map phase**
  - Each block of data is assigned to a map worker
  - The worker applies the **map function** to it
  - All intermediate results are sorted locally on the mapper
  - Once the map function finishes its work, the intermediate results are sorted on local disk of mapper
  - Also the intermediate results there are partitioned into R groups (partitioning is typically done by hashing) - these groups will be used at the reduce phase
- It needs to wait until *all* map tasks are completed
- Before the reduce phase starts
  - The Name Node assigns reduce task to idle workers (reducers)
  - The intermediate results are shuffled and assigned to reducers
  - Each reduces pulls its partition from mapper's disk (all map results are already partitioned by mappers)
- The **reduce phase**
  - Each reducer applies the **reduce function** to its own partition of data
  - The produced result is stored and replicated three times to ensure fault-tolerance.

Fault-tolerance is naturally achieved in this execution scheme: if a node is failed, the name node just re-assigns a task to another node in the cluster. Also it does not prepare any execution plan beforehand: once a task is done, the node is assigned to another task. This way it also achieves load balancing. Additionally, there are no communication costs between the nodes.

1.4 Advantages and Disadvantages

As a platform for data processing, Hadoop has many advantages:

- The MapReduce paradigm is quite simple, yet expressive; and it is especially suited for computing aggregated values.
- It is fault-tolerant,
- There is no dependency on data model or schema, which makes it especially attractive for handling unstructured data.
- It is flexible: MapReduce programs can be written in nearly any programming language
- It is cheap and runs on commodity hardware; plus it is open-source.

But also there are quite a few disadvantages of this model:

- There is no declarative high-level language such as SQL
- There are some performance issues
- The name node is a single point of failure

Let us discuss these disadvantages and how they are addressed.

There is no declarative high-level language such as SQL. MapReduce, although flexible, is quite a low-level approach and requires manual coding in some imperative programming language like Java or Python. This makes queries harder to write and maintain, while programmers that can do that are expensive. Especially this is bad for ad-hoc analytical querying. But there are possible solutions to this problem that introduce high-level SQL-like languages that compile into a series of MapReduce jobs. The most popular tools for this are Pig and Hive, and we discuss Hive in the section below.

There are performance issues. The most prominent problem is that both Map and Reduce operations are blocking: the reduce phase cannot begin until the map phase finishes. This apparently causes performance degradation which is especially noticeable for OLAP querying. But there are solutions for this problem, and one of them is Incremental MapReduce. We refer to [1] for further discussion on this problem. Additionally since there are no data model or schema, the input needs parsing, and therefore there are no indexes. The solution for this is to use HBase [13] or other database management system that works on top of HDFS.

Also Hadoop is bad in High Availability: there is a single point of failure, namely, the name node. This problem is typically overcome by buying special fault-tolerant hardware for the name node and regularly doing back-ups.

Finally, Hadoop is a very young technology, and it does not have decades of development behind and Relational Databases

2 Hadoop as a Data Warehouse

We see that Hadoop is proven useful with Big Data challenges and it is especially good for handling unstructured data such as free text, audio, video, or machine-generated data from sensors. What is mode, today the amounts of structured data grows rapidly and it becomes more and more expensive to keep all the data in one data warehouse.

There are companies that build their Data Warehouses entirely on Hadoop. In this section we discuss how it is possible to accomplish that and what are the advantages and disadvantages of this approach.

One of the successful examples is a system called “Cheetah” [6] used in a company Turn.inc. They provide special purpose marketing-oriented solutions and do that on top of Hadoop.

Inside they use typical data warehouse Show Flake and Star schemas with the central fact table. For being able to do that they introduce a concept of Virtual Views. A *virtual view* consists of columns that are exposed to users for querying, and inside they denormalize everything so the records inside are fully self-contained and no joins are needed. The denormalization strategy works well since all used dimensions are either append-only or slowly changing dimensions. It is not possible to query anything other than the columns specified by the virtual views and this is done for performance reasons.

Unfortunately this system is proprietary and it is not possible to install and try it. What is more, their approach is too specific and targeted only to the domain of web marketing

But Hive is a general purpose Data Warehousing solution [5]. This system was initially developed internally by Facebook, but now this is a widely used open-source technology for querying data that works on top of Hadoop.

Firstly, there is a Data Model in Hive that includes primitives, collections and user-defined types. The most basic structures are tables (like tables in Relational databases), each table corresponds to a directory in HDFS; partitions that divide tables; and buckets that divide partitions.

Additionally, there is a System Catalog where it keeps statistics and there is a query optimizer. However, all results are still materialized because of the inherited limitation of MapReduce: each MapReduce job needs its input stored on HDFS - therefore pipelining techniques are not possible.

Hive has a SQL-like declarative language for querying data that is called HiveQL. Consider the following example of a HiveQL query (taken from [5]). Suppose we have two tables: `STATUS_UPDATE(user_id int, status string, ds string)` and `PROFILES(userid int, school string, gender int)`. The column `ds` in this example is used to store date.

To load data we use the following command:

```
LOAD DATA LOCAL INPATH 'logs/status_updates'
INTO TABLE status_updates
PARTITION (ds='2009-03-20')
```

Note that we partition data over the date field.

Next, we want to compute daily statistics on how often a status is updated based on gender and school:

```
FROM
(SELECT a.status, b.school, g.gender
 FROM status_updates a JOIN profiles b
 ON (a.userid = b.userid and a.ds = '2009-03-20') subq1

-- groups by gender
-- inserts the result into another table
INSERT OVERWRITE TABLE gender_summary
PARTITION (ds='2009-03-20')
SELECT subq1.gender, count(1)
GROUP BY subq1.gender

-- groups by school
INSERT OVERWRITE TABLE school_summary
PARTITION (ds='2009-03-20')
SELECT subq.school, count(1)
GROUP BY subq1.school
```

Note that this is one query but with two subqueries: they are performed in a single scan.

Consider another query: suppose we want to list top-10 memes per school. Even though some constructs are impossible or hard to express via HiveQL, it is allowed to attach a custom external script to do the processing.

```
REDUCE subq2.school, subq2.meme, subq2.cnt
-- using custom python script
```

```
USING 'top10.py' AS (school, meme, cnt)
FROM (
  SELECT subq1.school, subq1.meme, count(1) as cnt
  FROM
  (MAP b.school, a.status
    USING 'meme_extractor.py'
    AS (school, meme)
    FROM status_update a JOIN profiles b
    ON (a.userid = b.userid)) subq1
  GROUP BY subq1.school, subq1.meme
  DISTRIBURE BY school, meme
  SORT BY school, meme, cnt desc)
) subq2
```

This query language along with Hive itself becomes increasingly popular.

As we see these approaches are either too specific or designed to handle semi-structured or unstructured data. However the majority of enterprizes deal with transactional data, for which there is no better solution than Relational Databases with almost half a century of development and research behind them. That is why usually Hadoop and Data Warehousing compliment each other, and not used alone. In the next section we discuss the interaction between them.

3 Hadoop and Data Warehousing

Hadoop plays a key role in capturing and aggregating raw data, it is cheap and scalable. But it cannot fully address the needs of Business Intelligence systems for building the reports based on traditional enterprize transactional data. Thus Hadoop and Data Warehouses should co-exist in one environment and used side-by-side, complimenting each other.

In this sections we see in what ways it is beneficial to use Hadoop and consider several use-cases.

3.1 ETL platform

The first and initial use-case of Hadoop in the Data Warehousing world is to use it as a transitory ETL platform. However Hadoop is not a replacement for any ETL tools, but rather just another component in them, and many vendors allow to plug in Hadoop with custom Hive and Pig queries.

The typical usage of Hadoop in a Data Warehouse is:

1. **E**xtract: load data into Hadoop's HDFS, parse data and prepare
2. Run some analysis on this data and try to find meaning and patterns
3. **T**ransform: clean the data and, with MapReduce, transform to some structured format
4. **L**oad: extract data from Hadoop and load into a Data Warehouse

So input in this case is some non-structured, machine-generated or semi-structured data. This kind of data is usually analyzed in isolation and doesn't need integration, unlike transactional operational data that is typically stored in Data Warehouses.

The most common usage of Hadoop as an ETL (according to [8]) is Text-Processing (and Pre-processing). Relational Databases are not good for this: there are no SQL functions for text-processing. Input in this case is some text stream, typically from social networks such as Twitter and Facebook, but it can also be blogs and web pages. This textual data in then analyzed inside a Hadoop cluster with to extract keywords or for sentiment analysis. Finally the processed data and extracted meaning are put into a Data Warehouse for further analysis, for example, to match sentiment with customer profile.

Another interesting use case is call-center monitoring. In a big company a typical call center receives too many phone calls with complaints. A solution to handle this data can be apply voice recognition algorithms, extract keywords and link them to the customers profile. In both cases the this information can later be used in many ways, one of which can be predicting customer churn.

In another example let us suppose that we run an e-commerce web site such as EBay, where customers post their advertisements and other customers buy. A customer publishes an advertisement for selling a red dress, but she does not mention the color (because it is apparent from the picture). However we want to be able to answer queries like "red dress". This, this image is processed and important features are extracted. This is then put not only to the operational database, but also to the data warehouse to be able to answer BI questions like what type of dress is most popular. This can be used to discover behavioral patterns of the customers.

3.2 Active Storage

Hadoop with HDFS can store many gigabytes of data. And this storage is very cheap: it costs 20 times less compared to complex and very expensive data management solutions [15]. This makes it a perfect solution for long-term data storage. What is more, with Hadoop it is possible to retrieve data fast and do some data analysis there and not to move everything back to a Data Warehouse each time we want to analyze data. Which is why Hadoop is sometimes called an "active storage".

For example, we keep all the call center recordings in Hadoop and do not put them into a Data Warehouse. But when needed, we may quickly run some queries over this set, for instance, to retrieve new keywords.

Another use case is to archive traditional transactional enterprize data. The amount of data that is not used in a DW grows over time. This is called "dormant" or "cold" data and can make up to 80% of all data [16]. We can move this data to a low-cost Hadoop storage which can results in extraordinary savings.

3.3 Analytical Sandboxes

Since the input data is not structured, it is often the case that analysts do not know what exactly they want to derive from data. This is another common use-case of Hadoop: load data into hadoop and try to analyze it and see what is there. Tools such as Hive make Hadoop a perfect match for this task.

Conclusions

The MapReduce computational paradigm has been becoming more and more popular in the today's world with petabytes of data around us. This paradigm, along with Hadoop as the most widespread implementation, have already proved useful in many areas, most prominently in social networks analysis as well as text processing and indexing. It has also been recognized as a convenient tool for Data Warehouses as well. We saw that it is possible to build a large data warehouse only based on Hadoop and the technologies that surround it such as Hive.

However, Hadoop is still quite far from reaching the maturity and performance of Data Warehousing solutions built on top of high-cost parallel relational database systems. But in spite of this fact these systems can effectively co-exist and compliment each other. The areas where Hadoop is the best include analysis of raw and semi-structured data, and this power is used during ETL stages, where Hadoop acts as a component in ETL flows. Once data is processed with MapReduce, it then can be utilized in a Data Warehouse by Business Intelligence tools.

Additionally due to its low cost it is also used as an active storage for dormant enterprize data and enables to retrieve and analyze this data quickly and efficiently.

It is clear that in the future this paradigm will be continuing to grow in popularity and maturity, and there will be more and more use cases where it fits. What is more, according to surveys (such as [8]) many companies that have not adopted Hadoop yet plan to do it within several years. That makes Hadoop a notable technology to become familiar with.

References

[1] Lee, Kyong-Ha, et al. “Parallel data processing with MapReduce: a survey.” *ACM SIGMOD Record* 40.4 (2012): 11-20. [pdf]

[2] “MapReduce vs Data Warehouse”. Webpage, `http://www.cbsolution.net/techniques/ontarget/mapreduce_vs_data_warehouse`. Accessed 15/12/2013.

[3] Ordonez, Carlos, Il-Yeol Song, and Carlos Garcia-Alvarado. “Relational versus non-relational database systems for data warehousing.” *Proceedings of the ACM 13th international workshop on Data warehousing and OLAP*. ACM, 2010. [pdf]

[4] A. Awadallah, D. Graham. “Hadoop and the Data Warehouse: When to Use Which.” (2011). [pdf] (by Cloudera and Teradata)

[5] Thusoo, Ashish, et al. “Hive: a warehousing solution over a map-reduce framework.” *Proceedings of the VLDB Endowment* 2.2 (2009): 1626-1629. [pdf] (by Facebook)

[6] Chen, Songting. “Cheetah: a high performance, custom data warehouse on top of MapReduce.” *Proceedings of the VLDB Endowment* 3.1-2 (2010): 1459-1468. [pdf]

[7] “How (and Why) Hadoop is Changing the Data Warehousing Paradigm.” Webpage `http://tdwi.org/articles/2013/08/13/hadoop-changing-dw-paradigm.aspx`. Accessed 15/12/2013.

[8] P. Russom. “Integrating Hadoop into Business Intelligence and Data Warehousing.” (2013). [pdf]

[9] M. Ferguson. “Offloading and Accelerating Data Warehouse ETL Processing Using Hadoop.” [pdf]

[10] Dean, Jeffrey, and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters.” Communications of the ACM 51.1 (2008): 107-113. [pdf]

[11] “What is Hadoop?” Webpage `http://escience.washington.edu/get-help-now/what-hadoop`. Accessed 15/12/2013.

[12] Apache Hadoop project home page, url: `http://hadoop.apache.org/`.

[13] Apache HBase home page, `http://hbase.apache.org/`.

[14] Apache Mahout home page, `http://mahout.apache.org/`.

[15] “How Hadoop Cuts Big Data Costs” `www.informationweek.com/software/a/d/d-id/1105546`. Accessed 05/01/2014.

[16] “The Impact of Data Temperature on the Data Warehouse.” whitepaper by Terradata (2012). [pdf]