Exploring the Efficiency of Big Data Processing with Hadoop MapReduce

Brian Ye, Anders Ye

School of Computer Science and Communication (CSC), Royal Institute of Technology KTH, Stockholm, Sweden

Abstract. Processing Big Data has throughout history always been a challenge for scientists, in both the academia and the industry. Hadoop MapReduce is a commonly used engine used to process Big Data. The Hadoop MapReduce framework uses a distributed file system to read and write data, called the Hadoop Distributed File Systems. It is also assumed that the software in the framework is reliable and fault tolerant. As the authors of the related paper state, Hadoop MapReduce often lack of appropriate indexes. Hadoop MapReduce has a performance problem due to its data layout and indexing, we want to combine an appropriate indexing technique and data layout of the database with Hadoop MapReduce in order to increase job performance and query processing. This project will take a theoretical approach since it is hard to implement an application considering the size of the data set that will be handled.

1 Introduction

Big Data is a concept that nowadays has become more relevant in current industries as well as academia. It has become a deciding variable for research and identifying approaches in terms of commercial and business development. In correlation with the current growth of information technology, and the trends established by the Internet, managing and processing information is important in order to make more accurate conclusions and analyses. These analyses can be based on for example user behavior, browsing patterns, and relating objects in social networks . As a result, a range of different methods and frameworks have been developed in order to be able to process and manage Big Data. To mention a few, cloud computing, clustering, and distributed Big Data platforms have emerged [2]. This report will be focusing on a distributed big data platform called *Hadoop MapReduce* by identifying its efficiency of processing Big Data with efficient usage of data layouting and indexing.

2 Background

MapReduce is a concept that has lately gained a lot of interest from both the database and systems research communities. Researches have started looking for other alternatives than using DBMS for processing data. As of now there exists many different forms of database processing methods, where DBMS techniques have been combined with MapReduce, and where only MapReduce functionality is used for processing Big Data [6]. As of this paper, the Hadoop MapReduce has been examined. Below is a figure showing the work-flow, from loading in the data to outputting the processed reduced data.

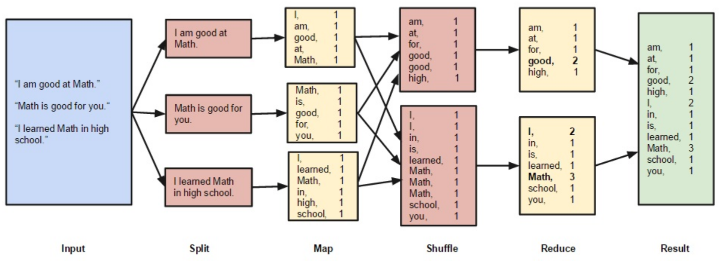
"I am good at Math."
"Math is good for you."
"I learned Math in high school."
I am good at Math.
Math is good for you.
I learned Math in high school.
I, 1 am, 1 good, 1 at, 1 Math, 1
Math, 1 is, 1 good, 1 for, 1 you, 1
I, 1 learned, 1 Math, 1 in, 1 high, 1 school, 1
am, 1 at, 1 for, 1 good, 1 good, 1 high, 1
I, 1 I, 1 in, 1 is, 1 learned, 1 Math, 1 Math, 1 Math, 1 school, 1 you, 1
am, 1 at, 1 for, 1 good, 2 high, 1
I, 2 in, 1 is, 1 learned, 1 Math, 3 school, 1 you, 1
am, 1 at, 1 for, 1 good, 2 high, 1 I, 2 in, 1 is, 1 learned, 1 Math, 3 school, 1 you, 1
Input
Split
Map
Shuffle
Reduce
Result

Fig. 1. Flow chart of MapReduce procedure

In order to understand Hadoop MapReduce, one must first examine MapReduce's functionality. MapReduce is a framework using a large set of commodity machines, known as a large cluster if all machines are in the same network, in order to solve parallelizable problems to solve Big Data sets [3]. The whole processing can be divided into three different steps, Map, $Shuffle$, and *Reduce*. In the illustration above, there is another step called *Split* which basically splits up the input data into smaller more manageable blocks. These blocks can be seen as the different commodity machines in the MapReduce process.

As for the Map step, a function $map()$, specified by the user, is applied on each machine and executes whatever the user has specified. The mapping works in such a way that there is a key mapping to a $value$. Specifically in the above figure, the frequency of each word is mapped.

As for the $Shuffle$ step, based on the output format the data is redistributed to fit this format. This step uses directly after what $map()$ will output. In this case, all words beginning with a to h is distributed in one block and the rest in another. Most importantly is that all data belonging to the same key exists in the same block.

As for the *Reduce* step, the data for each block, and each key, is processed in parallel.

Hadoop MapReduce is an engine used to process Big Data, and the largest reason Hadoop is widely used is because it has a simple interface for the programmer and yet comes with the powerful features a MapReduce process provides. The only two tasks needed to be done manually by the programmer is to define the $map()$ and the $reduce()$. Another property making Hadoop MapReduce unique in its way is that it uses the Hadoop Distributed File System (HDFS) for storage and is what later processed in MapReduce. With this said, Hadoop splits files into large blocks which are distributed over a large cluster and prepared to be processed [9]. However, even though Hadoop seems to be a good alternative for processing Big Data some drawbacks still need to be considered. It is known that Hadoop MapReduce lacks index access to the provided data, because priori knowledge of the MapReduce jobs being executed is not known. Furthermore another property that Hadoop MapReduce has, which is more of a restriction rather than a drawback, is that the data being processed are labeled in rows thus having a row layouting of the data [6]. These characteristics will later be brought up, where we will identify the advantages as well as disadvantages of having such data layout in combination with other layouting techniques.

3 Index

Indexing is one of the most important methods in terms of optimizing query processing and fetching data from database systems. There is a selection of different indexing techniques, but all in all they have the purpose to tune an existing database. The drawbacks with indexing comes with that disk space must be allocated for the index data structure, as well as writes becoming more expensive. Queries consisting of $INSERT$ or $UPDATE$ will also make the whole index data structure to update as well, in addition to updating the database with the newly written values [5].

3.1 Hadoop++

Hadoop++ is an alternated version of the original Hadoop implementation, which makes it fundamentally the same as the original one though with some important adjustments. A covering index is used on each split data block, as illustrated in the flow chart above. A covering index is an index such that the index itself contains the designated data attributes, thus speeding up retrieval time even more. Before executing the MapReduce jobs the data has already been indexed since the indexes are created during upload time. It is as a result known that query processing is up to 20 times faster than regular Hadoop [6]. It is however also observed that the upload time will be longer, which is due to that the index structure needs to be processed as well.

3.2 HAIL

HAIL (Hadoop Aggressive Indexing Library) is an enhancement of Hadoop improving both upload and run time. It improves upload time by using binary PAX layout which decreases the size of the data. This will be further discussed in coming section. In order to improve run time, different clustered indexes are used on each replica in order to increase the likelihood to find a suitable index for a MapReduce job. Clustered indexes sorts the data rows and stores them based on the key, which is why only one clustered index can exist per replica [8].

4 Data Layout

Data layout(ing) is a core concept in modern database systems. Since different data layout have different pros and cons, it is therefore important to identify the needs for the database and then choose a corresponding data layout.

4.1 Row Oriented vs. Column Oriented Layout

In row oriented layout the data is stored in sections of rows. This means that the data is stored record by record, so if a record is found we can easily access all the attributes for that record. The drawback to this kind of layout is the analytics. It takes a lot of time to access a single attribute for all records [4].

In column oriented layout the data is stored in sections of column. The data is stored attribute by attribute, even if we found an attribute for a record we will still need to search for the other attributes for that record. But column oriented layout is very good for analytics since the attributes are stored together [4].

To illustrate the difference between these techniques we have the following example. Assume we have a database with information of all students attending KTH. If we wanted to know the age and department of a specific student, it would be appropiate to use the row oriented data layout. If we find the student in the database we can then easily retrieve the desired data. Though if we instead wanted to know the average age of all the students then we would need to access the age attribute of all students, which means that the column oriented layout is more effective.

The row oriented layout is good for multiple attribute access and bad for analytics. The column oriented layout is bad for multiple attribute access and good for analytics [4].

4.2 PAX Layout

PAX (Partition Attributes Across) layout only affects the data stored on a single page, which means that it can be combined with other data layout techniques. The PAX layout was introduced to minimize accesses to main memory by getting less cache misses. PAX groups all values of each attribute in mini-pages, so when we search for a value, lots of values are loaded at the same time to the cache. Also the whole record is on the same page, so only mini-joins among mini-pages are needed to get the whole record [1].

Binary PAX Layout Binary PAX Layout uses PAX layout but at the same time converts the data into binary form in order to reduce the size of the data. This results in less I/O and therefore saves time during upload [8].

Trojan Layout The Trojan Layout was inspired by PAX Layout. It only changes how the data is stored on a single page and not between the pages. However the Trojan Layout do co-locate attributes together depending on query-workloads and also uses different page layout for the different replicas [10].

5 Further Studies

The Trojan Layout still has room for improvements. Further research on how attributes can be co-located depending on query-workloads can be made. In addition to that, research can be made on what kind of page layout should be used for the different replicas in order to maximize run time speed.

Both Hadoop++ and Hail uses indexes to improve the run time speed for Hadoop. However if we could improve the index selection algorithm for each indexed replica, the performance of both implementations can be improved.

6 Conclusion

Indexing is a way to improve job performance, though one should also manage the upload time so it does not affect the overall performance of the system. In order to still have low upload time, it is important to provide appropiate data layouting techniques. Using a good data layout does improve the run time for Hadoop. However different layouts are good for different cases, so it is important to choose a data layout according to what is specified by the end-user.

References

1. Ailamaki, A., DeWitt, D.J., Hill, M.D., Skounakis, M.: Weaving relations for cache performance. In: VLDB. vol. 1, pp. 169–180 (2001)
2. Davenport, T.H., Dyché, J.: Big data in big companies. May 2013 (2013)
3. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
4. Dittrich, J.: 14.216 data layouts: Row layout vs column layout (2014), https://www.youtube.com/watch?v=I-jnZ22r7fY
5. Dittrich, J.: Insert performance with indexes (2014), http://use-the-index-luke.com/sql/dml/insert
6. Dittrich, J., Quiané-Ruiz, J.A.: Efficient big data processing in hadoop mapreduce. Proceedings of the VLDB Endowment 5(12), 2014–2015 (2012)
7. Dittrich, J., Quiané-Ruiz, J.A., Jindal, A., Kargin, Y., Setty, V., Schad, J.: Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing). Proc. VLDB Endow. 3(1-2), 515–529 (Sep 2010), http://dx.doi.org/10.14778/1920841.1920908
8. Dittrich, J., Quiané-Ruiz, J.A., Richter, S., Schuh, S., Jindal, A., Schad, J.: Only aggressive elephants are fast elephants. Proceedings of the VLDB Endowment 5(11), 1591–1602 (2012)
9. IBM: What is the hadoop distributed file system (hdfs)? (2014), http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/
10. Jindal, A., Quiané-Ruiz, J.A., Dittrich, J.: Trojan data layouts: right shoes for a running elephant. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. p. 21. ACM (2011)
11. Jindal, A., Quiané-Ruiz, J.A., Dittrich, J.: Trojan data layouts: right shoes for a running elephant. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. p. 21. ACM (2011)