Intentional Data Placement Policy for Improving OLAP Cube Construction on Hadoop Clusters

Billel Arres, Nadia Kabachi, Omar Boussaid

Intentional Data Placement Policy for Improving *OLAP* Cube Construction on Hadoop Clusters

Billel ARRES
2nd year PhD student
Universite Lumiere Lyon 2
b.arres@univ-lyon2.fr

Nadia KABACHI
Advisor
Universite Lyon 1
n.kabachi@univ-lyon1.fr

Omar BOUSSAID
Director
Universite Lumiere Lyon 2
o.boussaid@univ-lyon2.fr

ABSTRACT

In the recent past, we have witnessed dramatic increases in the volume of data literally in every area: business, science, and daily life to name a few. The Hadoop framework - an open source project based on the MapReduce paradigm - is a popular choice for big data analytics. However, the performance gained from Hadoop's features is currently limited by its default block placement policy, which does not take any data characteristics into account. Indeed, the efficiency of many operations can be improved by a careful data placement, including indexing, grouping, aggregation and joins. In our work we propose a data warehouse partitioning strategy to improve query gain performances. We investigate the performance gain for OLAP cube construction with and without data organization on a Hadoop cluster. And this, by varying the number of nodes and data warehouse size. Our experiments suggest that a good data placement on a cluster during the implementation of the data warehouse can significantly increase the OLAP cube construction and querying performances. In the next step, we will extend the experiments to study the effects of other configuration parameters on collocation data in the context of parallel data warehousing, such as partitions size, replication factor and OLAP query complexity. We plan also to study an intelligent system for warehouses data placement on clusters by integrating Multi-Agent System (MAS) and Intelligent Agents to the process.

Keywords

MapReduce, HDFS, Data warehouses, Block Placement

1. RELATED WORK

In this section, we describe the MapReduce paradigm in the context of OLAP. We then discuss the Hadoop data organization enhancement techniques in brief.

1.1 MapReduce and OLAP

BDA 14 octobre 2014, Grenoble-Autrans, France.

MapReduce [9] is a framework for parallel processing of massive data sets. A job to be performed using the MapReduce framework has to be specified as two phases: the map phase as specified by a Map function, takes key/value pairs as input, performs some computation on this input, and produces intermediate results as key/value pairs; and the reduce phase which processes these results as specified by a Reduce function. The data from the map phase are shuffled, i.e., exchanged and merge-sorted, to the machines performing the reduce phase. It should be noted that the shuffle phase can itself be more time-consuming than the two others depending on network bandwidth availability and other resources.

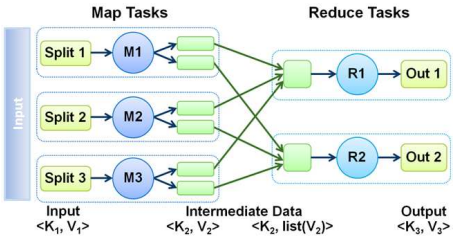

Figure 1: The MapReduce model.

MapReduce runs in cluster of nodes; one node acts as a master node (called Namenode) and other nodes act as workers (called Datanodes). It efficiently uses network bandwidth by moving computation to data. The input data is managed by a distributed file system[11] which divides input data into a set of blocks; the block size can be specified by users. In addition, it replicate each block, the default replication number is three, and puts one replica in the same rack and puts the other replica in other rack. The strategy of replica distribution helps in restoring data in case of node or rack failures. The MapReduce programming model has many advantages, such as high throughput/performance, use of commodity machines, and fault tolerance. Hence, MapReduce is used in not only index construction for search engines [9] but also data analysis of both homogeneous and heterogeneous sets [7]. Data join processing, which is very important for complex analysis in data warehouses, is addressed in [7] and [10] using MapReduce. Other works like [3] and [6] used Hadoop-based implementations, such as Hive [2] and CloudBase [6], as alternatives to relational DBMSs benchmarking and comparing different approaches to retrieve OLAP data cubes with MapReduce.

In fact, MapReduce can be useful for OLAP processing in large data warehouses. For example, Facebook has implemented a large data warehouse system using MapReduce instead of DBMS's [5]. According to [8], a data warehouse is an online repository for data from operational systems of an enterprise. It is usually maintained using a star schema that is composed of a single fact table and a number of dimension tables. A fact table contains atomic data or records for business areas such as sales and production. Dimension tables have a large number of attributes that describe records of the fact table. In the context of MapReduce all data in data warehouses are stored as a form of chunks in distributed file system [9]. The data warehouse is splited into *blocks* and distributed over the cluster nodes. However, the MapReduce data file system tries to balance the load by placing blocks randomly, independently of the intended use of the data. In this paper, we focus on the performance gain by careful data organization for star schema data warehouses.

1.2 The Hadoop Framework

Hadoop [12] is a MapReduce based framework designed for scalable and distributed computing. It consists of two main parts: the Hadoop distributed file system (HDFS) and MapReduce for distributed processing. Files in HDFS are split into a number of large blocks (usually a multiple of 64 MB) which are stored on Datanodes. A file is typically distributed over a number of Datanodes in order to facilitate high bandwidth and parallel processing. In fact, efficient access to data is an essential step for achieving improved performance during query processing which is a very important step in data warehousing context. Indeed, in contrast to many positive features of MapReduce and it's open-source implementation Hadoop, such as scalability and fault tolerance, it has some limitations, especially in data access. According to literature [1, 4], three subcategories of data access improvement exists, namely indexing, data layouts, and intentional data placement.

2. PROBLEM STATEMENT

A data warehouse is an online repository for data from operational systems of an enterprise. It is usually maintained using a star schema that is composed of a single fact table and a number of dimension tables. In the context of MapReduce all data in data warehouses are stored as a form of a chunk in distributed file system. The data warehouse is split into *blocks* and distributed over the cluster nodes. However, the Hadoop Data File System tries to balance the load by placing blocks randomly, independently of the intended use of the data. Our study focuses on the performance gained by careful data organization for star schema data warehouses. Assuming a star schema data warehouse with a fact table FF and four dimensions $D1$, $D2$, $D3$, $D4$. With the Hadoop distributed file system HDFS, all the data in the data warehouse are split into blocks of fixed size and stored on Datanodes. The block size is configurable and defaults to 64MB. So there will be FF_i, $D1_j$, $D2_k$, $D3_m$ and $D4_n$ blocks in the file system. The integers $\{i, j, k, m, n\}$ depends on the size of each table.

By default, the data placement policy of HDFS tries to balance load by placing blocks randomly on the Datanodes (Fig. 1). This default data placement policy of HDFS arbitrarily places partitions across the cluster so that mappers often have to read the corresponding partitions from remote nodes. In this case, the network overhead can be eliminated or at least reduced by colocating the corresponding partitions, i.e., storing them on the same set of Datanodes as shown in Figure 2. Furthermore, it improves the efficiency of many operations such as joins and grouping.

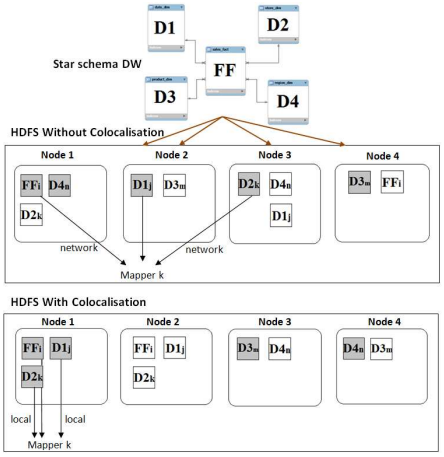

Figure 2: DW blocks processing without colocation vs. with colocation.

3. SYSTEM IMPLEMENTATION

The main idea of our work suggests that to improve data warehouse query performances, particularly OLAP queries, we must first define a good strategy for data distribution. In this case, we propose to colocate on the same set of Datanodes the fact table partitions, related attributes and dimensions partitions which are involved in the user query. We implement our data warehouse approach by using the HDFS-385 API (Version 1.2.0 released on May 2013) which is an expert-level interface for developers who want to try out custom placement algorithms to override the HDFS default placement policy. We used also a data placement mechanism called locator (M.Eltabakh, et Al. 2011). During the loading phase, each data warehouse table file is assigned to at most one locator and many tables files can be assigned to the same locator. Tables data files and partitions with the same locator are placed on the same set of Datanodes, whereas others with no locator are placed via Hadoop's default strategy. The table locator is set with default values according to the policy location initially defined.

Figure 3 shows the locator table corresponding to four tables files colocation on a cluster. Our approach was evaluated with well-known, large-scale data analysis benchmark: SSBM (Star Schema Benchmark). It is a data warehousing benchmark derived from TPC-H. The star-join query, in which the fact table is joined with one or more dimension tables, is one of the well-known queries in OLAP. In a context of parallelization and data colocation we have chosen to evaluate our approach with a star-join OLAP cube construction query that involve only two dimension tables.

4. EXPERIMENTAL EVALUATION

4.1 Experimental Setup

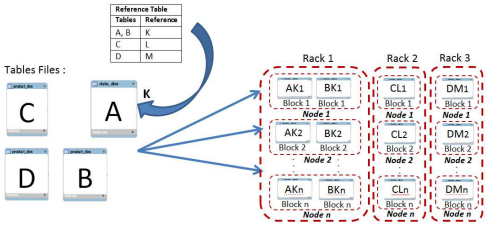

Figure 3: Example of four tables files colocated using locator on three nodes cluster.

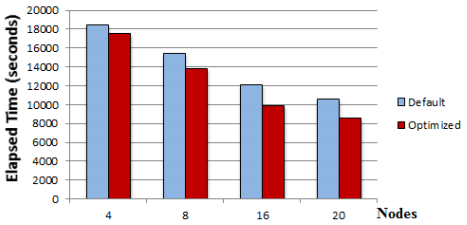

Figure 5: OLAP cube construction time by varying number of nodes. (DW size is 920GB)

In the experiments, we used a total of 20 PCs in a cluster. The operating system is Ubuntu 12.04 LTS, and the MapReduce framework is Hadoop 1.0.3. In the experiments, we compare the performances (Execution time) of the OLAP cube construction query presented previously, first, without optimization (Default) by using the Hadoop.1.0.3 version, then with optimization (Optimized) by using our HDFS extension with the same Hadoop version. We used Hive.0.10.0 for the execution of the query for both platforms.

4.2 Experimental Results

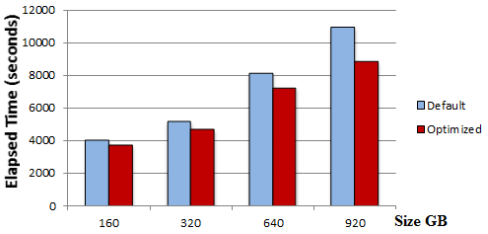

Figure 4: OLAP cube construction time by varying data warehouse size. (20 nodes cluster)

As shown in Figure 4, cube computation execution time increases significantly as the data size increases and the benefits of the proposed data warehousing colocation approach are appreciable with the increasing size compared to default HDFS data distribution. The Figure shows that tables files colocation improves the query performance from 7% for 160GB to 19% for a 920GB data warehouse size. This behaviour is expected since colocation of data, in the context of data warehousing, avoids network overhead, besides, it reduces the expensive data shuffling operation compared to default data placement policy.

In Figure 5 we observe that the execution time of cube computation decreases as the number of nodes increases. In contrast, colocating data improves query performance as the cluster getting larger. We also note that intentional data warehouse placement is more suitable for large clusters, this is because the map and reduce jobs avoids the data shuffling/sorting phase, moreover, reading the data locally is much faster than reading data over the network.

5. CONCLUSION

In this first step of work, we present a data warehouse partitioning strategy to improve query performances on a Hadoop cluster. By adopting existing colocation mechanisms, we empirically tested the performance gain for OLAP cube queries with and without data colocation.

6. REFERENCES

[1] A.Elsayed, O.Ismail, and M.E.El-Sharkawin. Mapreduce: State-of-the-art and research directions. *International Journal of Computer and Electrical Engineering*, 6(1):34–39, 2014.

[2] A.Thusoo, J.S.Sarma, N.Jain, Z.Shao, P.Chakka, S.Anthony, H.Liu, P.Wyckoff, and R.Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[3] B.Arres, N.Kabbachi, and O.Boussaid. Building olap cubes on a cloud computing environment with mapreduce. In *ACS International Conference on Computer Systems and Applications*, pages 1–5, 2013.

[4] C.Doulkeridis and K.Norvagl. A survey of large-scale analytical query processing in mapreduce. *The VLDB Journal*, 23(3):355–380, 2014.

[5] C.Monash. Cloudera presents the mapreduce bull case. *http://www.dbms2.com/2009/04/15/cloudera-presents-the-mapreduce-bull-case*, 2009.

[6] H.Han, Y.C.Lee, S.Choi, H.Y.Yeom, and A.Y.Zomaya. Cloud-aware processing of mapreduce-based olap applications. *Proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing*, 140:31–38, 2013.

[7] H.Yang, A.Dasdan, R.-L.Hsiao, and D.S.Parker. Map-reduce-merge: simplified relational data processing on large clusters. *SIGMOD '07*, 2007.

[8] W. H. Inmon. The data warehouse and data mining. *Communications of the ACM*, 39(11):49–50, 1996.

[9] J.Dean and S.Ghemawat. Mapreduce: simplified data processing on large clusters. *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[10] R.Pike, S.Dorward, R.Griesemer, and S.Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming - Dynamic Grids and Worldwide Computing*, 13(4):277–298, 2005.

[11] S.Ghemawat, H.Gobioff, and S.T.Leung. The google file system. *the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[12] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.