Parallel Processing in Data Warehousing and Big Data Analytics

Abhishek Manoj Sharma
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000
abhishekmanoj.sharma@sjsu.edu

ABSTRACT

Data and information generation is increasing at an exponential rate. This outburst of data has led to large computations for information retrieval and data analysis. On-line analytical processing (OLAP) data warehousing systems like Oracle have started adopting to parallel processing. They use a central system of a query coordinator which communicates between the client and the server. Similarly, dedicated Big Data systems like Hadoop are also built upon the core techniques of parallel processing [1]. Such systems use a set of steps relying on splitting and merging of tasks to achieve parallelism.

This paper studies the existing parallelism implementations in relational OLAP and Big Data systems. The project also compares the implementation in such systems and discusses the pros and cons of those techniques. Finally, the project discusses the architecture and design of the parallel processing techniques used in different database systems.

1. INTRODUCTION

The world presently is booming in data creation, and in this era of extensive data generation, it is important to retrieve meaningful information from a plethora of data. Therefore, the concepts of data warehousing, OLAP systems, and big data are of major importance to the industry. However, with this increase in data comes the challenge of processing this data efficiently. Additionally, in the competitive sectors, results are expected in a timely manner. This combination of having efficient data analytics and information retrieval in quick time has compelled major tech companies to invest in competent warehousing and analytics systems. These companies are now investing heavily to research on how a combination of software with correct hardware can enable to achieve this goal [2]. On the other hand, some companies have come with up their own proprietary solutions to analyze data using the modern parallel hardware architecture.

OLAP systems have started migrating to parallel computers and architecture for data analysis. OLAP is used for summarizing, synthesizing, and consolidating data to multiple dimensions. The kind of queries used in OLAP systems are complex and involve a huge amount of data. Hence, it is necessary to use optimal hardware solutions to maximize the use available for resources for this data processing.

Similarly, Big Data systems also deal with a large amount of data. The volume of data processed by these systems can be enormous, and the data handling techniques must be very efficient. On the other hand, these systems also assist in data mining, resulting in extensive concentrated searches over this massive data. The MapReduce programming model is considered one of the most optimal approaches for working with big datasets in a distributed and parallel environment. Building upon this, Apache's Hadoop has customized the MapReduce implementation, which is a more widely used approach from the industry's standpoint. The MapReduce model works in two phases – map and reduce. The map phase performs tasks like sorting or filtering, whereas the reducer is used more in the programming context. Oracle, on the other hand, has also introduced parallelism for data warehousing. Their implementation is the first of its kind for leveraging the advantages of multi-core processors.

2. PARALLEL EXECUTION

Most current database systems, irrespective of warehousing OLAP systems or On-line transaction processing (OLTP) systems hold a large amount of data. As this size of data increases, it becomes more important to optimize the data processing techniques. One solution to this problem is parallelism. Using parallel processing, even petabytes of data can be processed in few minutes. Thus, it significantly reduces the time taken in running queries on large systems.

To signify the importance of parallel execution, let us consider a task of counting the number of fruits in different baskets. One approach is to count the number of fruits in each basket sequentially. If counting the fruits in one basket takes 's' seconds, then for 'n' baskets the total time spent is 'n x s' seconds. However, if 'n' people count the number of fruits in 'n' baskets and then share their counts with each other, then the total time required reduces from 'n x s' seconds to just 's' seconds.

As the value of 'n' increases, this gap increases further, signifying the growing importance of parallelism. Most database systems make use of parallelism focusing mainly on speed of execution as the degree of time saved by parallelism increases with the increase in the amount of data.

An important thing to understand while implementing parallel processing is to understand when to implement parallelism and when not to. This is especially important in designing parallel systems as parallelism may not always be optimal. There are certain scenarios where sequential approach may be better than

parallel approach, and hence this knowledge becomes important while system designing.

Parallel processing must be implemented only when the system has clusters, symmetric multiprocessors, enough I/O bandwidth, sufficient memory, and intermittently used CPUs [3]. If any of these thing is missing, parallelism can degrade the performance instead of improving it.

Similarly, parallelism should be strictly avoided in environments where queries are generally short or when the hardware resources like memory, I/O, and CPU are utilized heavily.

Thus, parallel processing is implemented in the software but has to follow the correct hardware. Hence, it is important to have the right set of machines to achieve a high degree parallelism. To summarize it, parallelism is a software solution and a hardware challenge.

3. PARALLELISM IN OLAP SYSTEMS

OLAP is the fundamental foundation in different fields of data analytics. The OLAP server contains the query processing logic for each form. Using the multi-dimensional cubical view of OLAP server illustrated in figure 1, features such as roll-up and drill-down can be efficiently implemented. Most business intelligence activities require competent data reporting, often requiring decisive roll-up and drill-down functionalities.

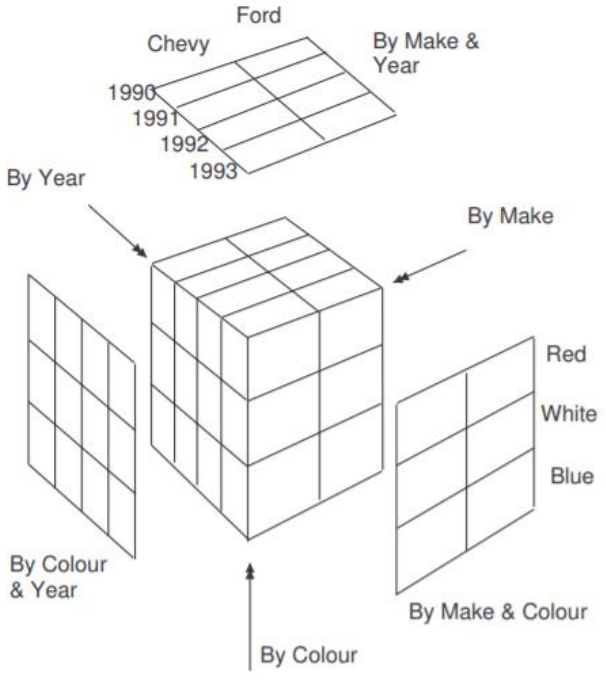

Figure 1. A three-dimensional OLAP cube representation

The OLAP Architecture follows a model where the data warehouse is connected to the OLAP servers and the data marts [4]. It is a three-tiered architecture illustrated in figure 2.

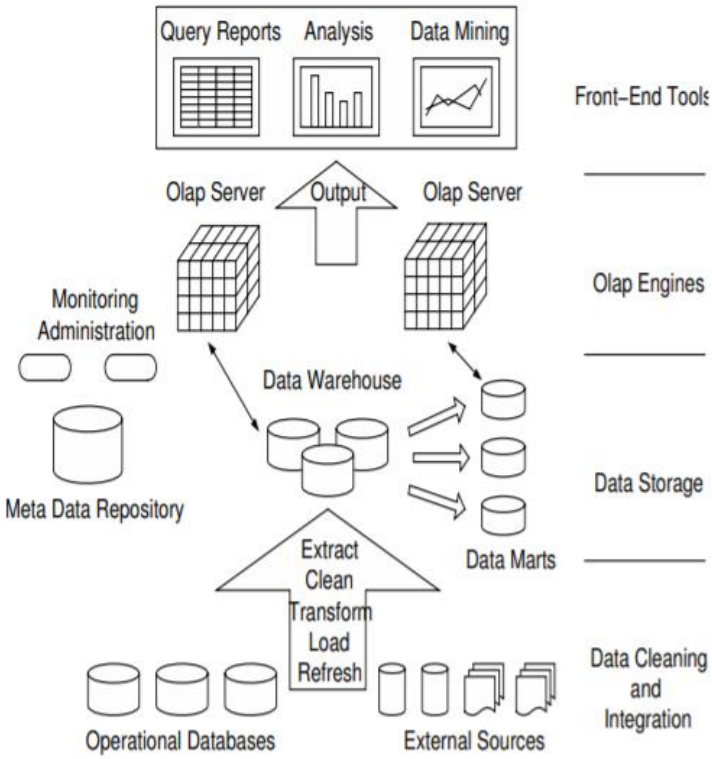

Figure 2. OLAP Mode- Three tiered

Oracle's implementation of parallelism in this model is heavily based on an architecture where the optimizer receives a SQL statement and determines the execution plan for it [4]. A threshold time is set, and the sequential execution plan's time is compared to the threshold plan. If the estimated time is less than the threshold time, then the statements are executed serially.

However, if the execution plan exceeds the threshold time, then an optimizer is used to determine the ideal degree of parallelism. The actual degree of parallelism is calculated as the lower of *PARALLEL_DEGREE_LIMIT* and the ideal degree of parallelism [5]. After determining the actual degree of parallelism, the statements are executed in parallel.

This process is known Automatic Degree of Parallelism (DOP). If DOP is implemented, many queries are observed to be running in parallel for lower thresholds. This is because the lower threshold is, easier is it to surpass the threshold [5].

Therefore, when a lower threshold is set, it is important to check the processor utilization in the system. A very high number of queries running in parallel can in theory exhaust the processors from performing any other task. A diagrammatic representation of automatic DOP is illustrated in figure 3.

The figure illustrates how the query is hard parsed by the optimizer for determining the execution plan and then comparing it with the threshold time of the system. Hence, the number of queries run in parallel largely depends on threshold time set in the system.

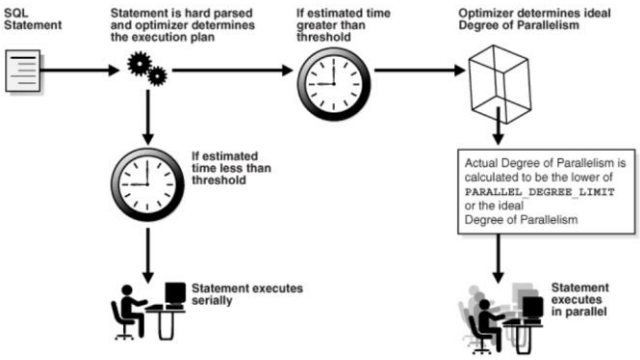

Figure 3. Automatic DOP in Oracle's OLAP System

3.1 IN-MEMORY PARALLELISM

Prior to Oracle 11g Release 2, all parallel executions scanned large tables using direct read mechanism. As a result, they completely bypassed the buffer cache to directly read and scan the tables [6]. As time progressed, the size of buffer cache has increased and hence in-memory parallel execution of queries (known as IMPQ) was introduced.

Parallel execution divides the objects in small granules for execution. Each granule is either a range, block, or partition depending on the object's physical definition. These granules are then assigned to the parallel execution servers for them to process it [6].

Figure 4 represents how the granules are formed as nodes. The upper half of the table is cached in node 1, and the lower half is cached in node 2. Both these nodes are then executed in a parallel manner. When another statement for parallel execution comes in, the parallel execution mechanism reads the data from the cache for faster execution.

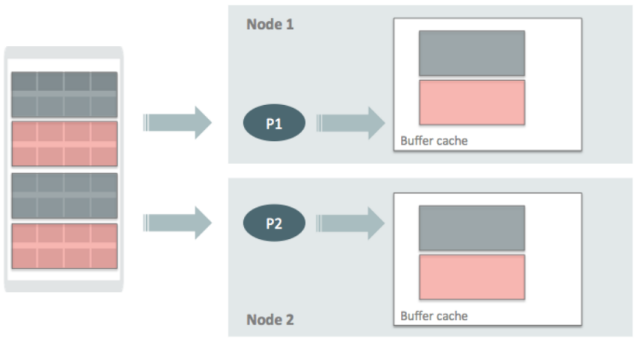

Figure 4. Representation of node formed from tables

In-memory parallel execution uses flags to decide the degree of parallelism. One such flag is *PARALLEL_DEGREE_POLICY* which helps Oracle DBMS decide if the query execution will be benefit from parallel execution [7]. If the IMPQ architecture is unable to decide a flag value, the default value of *AUTO* is set for the execution. In this case, the decision of executing parallelism lies on the information contained in the cache. This allows Oracle's OLAP system to be dynamic in terms of parallel execution instead of having a brute force approach of always executing in parallel without checking the execution plan.

4. PARALLELISM IN BIG DATA

Big data tools and frameworks work with massive amounts of data. Therefore, it is integral that the software makes optimal use of the hardware to run queries in a manner where maximum efficiency can be achieved. The MapReduce programming model is a widely adopted model for big data processing in parallel and distributed environments. It works on the principle of dividing the tasks in different phases, executing the steps inside those phases in a parallel way and then moving to the further phases. The two integral phases of the MapReduce model are the mapper and the reducer phases [8].

The mapper phase primarily takes as input a set of key-value pairs and processes them as needed. The output type of the mapper phase need not be same as the input type. The map model divides the data into chunks, then filters and sorts them, and provides its output as an input to the final reducer phase. Hence, the output of mapper is also said to be intermediate outputs. Additionally, the chunks formed during processing may be of different sizes based on the use case.

After the map phase, technically comes the phase of shuffle, sort, and merge. This is an optional phase unlike map and reduce. This phase acts as an intermediator between the mapper and the reducer. The output of the mapper phase goes to the reducer as an input via the intermediate phase. During this phase, the values are shuffled or sorted as per the requirement and are eventually merged based on keys [8]. Once this is complete, the reducer is invoked with this merged data.

Lastly, the reducer is where the logical program execution of the model exists. It reads all the values for every hashed key and performs a set of specific tasks on that data. The output from the reducer is generally a key-value pair. The data flowchart of the MapReduce model is illustrated in Figure 5.

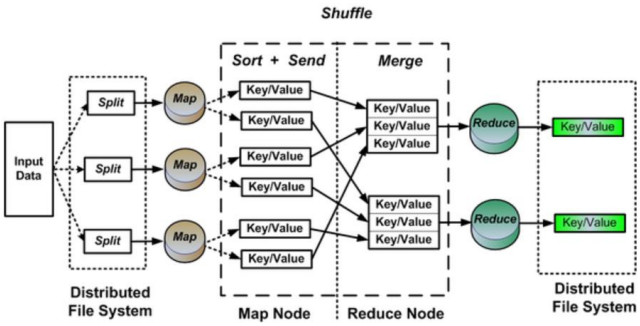

Figure 5. MapReduce model and architecture

4.1 APACHE HADOOP

Using the MapReduce programming, Apache has built their open-source big data computation framework called Hadoop. Hadoop is a collection of tools and utilities for facilitating computations on massive sets of data. The entire framework is built up on Java. The key features of Hadoop are task control, fault tolerance, task supervision, and multiple user management. It makes use of its own proprietary file system known as the Hadoop Distributed File System (HDFS), which is a derivation of the Google File System (GFS) [9]. The key components of developing HDFS over GFS were to incorporate distribution of redundant data over many

machines. Such machines are represented by nodes called DataNodes. The architecture used for replication and partitioning of these nodes is the master slave architecture [9]. The architecture is illustrated in Figure 6.

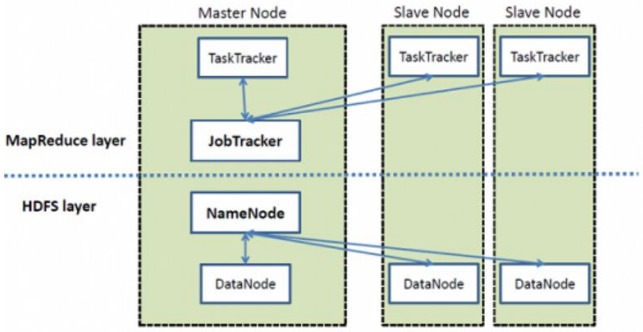

Figure 6. High - level Hadoop architecture

Every node, irrespective of it being the master or slave node is divided into two layers, a MapReduce layer and a HDFS layer. Additionally, each node consists of DataNodes and task trackers. However, these DataNodes and task trackers must point to a single NameNode and Job Tracker respectively. Both these components exist in the master node.

The task trackers and job tracker are present in the MapReduce layer of the master node, whereas the slave nodes only have the task trackers in their MapReduce layer. Similarly, the DataNodes and NameNode are present in the HDFS layer of the master node, whereas the slave nodes only have the data nodes pointing to the name node of the master node.

4.2 HDFS ARCHITECTURE

Every file present on a Hadoop Distributed File System is split into blocks. These blocks are then replicated over the network. The maximum size of a block can be as much as 256 megabytes, however, the default size set by HDFS design is 64 megabytes.

The filesystem uses separate servers for application and file data. All application related data is stored in the DataNodes and file data is stored in the NameNodes. HDFS replicates this data using a replication factor to maintain the reliability of the system. The HDFS architecture diagram is illustrated in Figure 7.

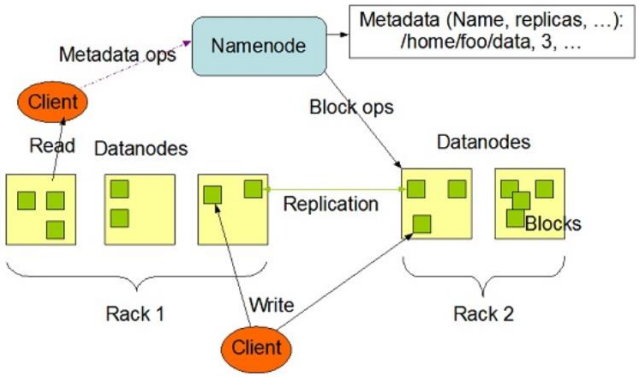

Figure 7. HDFS Architecture

A HDFS architecture is said to be efficient if it satisfies the below two conditions:

- Throughput of hard drives must be high.
- Network must be stable and have good speed for transferring data and replicating blocks.

4.2.1 NameNode

Every file in the HDFS present inside the NameNode is represented by Inodes. These Inodes contains information like timestamp, access methods, access times, and the mapping to the file system.

4.2.2 DataNode

A DataNode manages the current status of its node in the Hadoop Distributed File System. It is designed to carry out functions like extensive I/O tasks, clustering, batch processing, machine learning algorithms, compression, decompression, etc. Hence, a DataNode must have a lot of I/O for processing and transferring data.

4.3 APACHE SPARK

In the present world, most of the data exists from sources such as data warehouses, HDFS, non relational or relational database systems, etc. However, with the advancements in API based programming, generated data can be transferred instantly as streams. Some examples of this would be tweets from Twitter, commentary of a sporting event, and election vote counts. All of these scenarios have real-time data generated in them which can be streamed to a database system [10]. Apache Spark is built for these applications. Spark supports streaming data for implementing big data analysis on it.

Figure 8 illustrates the architecture of Apache Spark. It contains of a central cluster manager which connects to every other component in the architecture. There is a driver which connects to the cluster manager, which ahead connects to multiple workers in the system. Each worker contains an executor in them. The drivers and executors have their own independent Java processes and run them separately [10].

The Architecture consists of two primary components as abstractions:

1. Directed Acyclic Graph (DAG)

2. Resilient Distributed Dataset (RDD)

4.3.1 Directed Acyclic Graph (DAG)

It is a collection of programs and computations done on data. Every node denotes in an RDD partition, and every connecting edge represents the transformations run over it.

4.3.2 Resilient Distributed Dataset (RDD)

It is a collection of items that are divided in multiple partitions. These partitions can then be stored in the memory of the worker nodes. In a typical scenario, Spark RDD supports parallelized collection of data and Apache Hadoop datasets. Similarly, they

support performing operations like actions and transformations on each of them. The RDDs are thus an integral abstraction in the Apache Spark architecture.

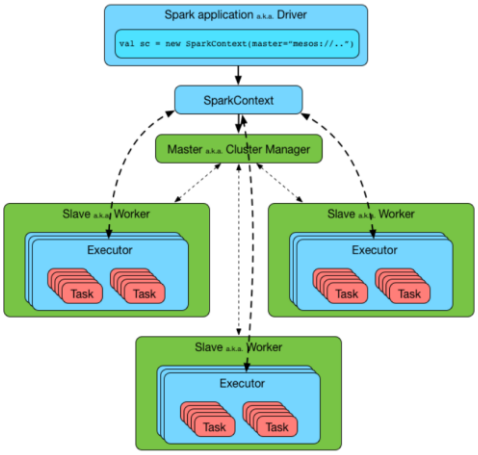

Figure 8. Apache Spark Architecture

The overall architecture, like Hadoop's architecture, is based on a master/slave system. The cluster manager acts as the master, whereas every worker acts as a slave. As discussed earlier, each slave has an executor which perform a set of designated tasks. The exit result of these tasks is sent to the SparkContext which is used by the Spark Application Driver to monitor the system. In this way, as the data keeps streaming in to the system, the execution of tasks in the executor and the data read process run parallel to each other. In case of an unexpected behavior, the error is sent to the SparkContext which is read by the driver to take an appropriate action depending on the type of error / warning and the availability of resources [10]. As a result, a good degree of fault tolerance is achieved in the complete streaming system.

5. CONCLUSION

Different kind of database systems are being used currently. If large volumes of data is taken into consideration, then the primary type of systems are OLAP warehousing systems and big data frameworks. Oracle has implemented some extremely efficient parallel processing mechanisms to implement parallelism and distribution on OLAP systems. They use the concept of degree of parallelism which can be dynamic based on the threshold time, and hence avoids the situation of having unnecessary parallelism when not needed. That is, it prefers sequential execution on simpler queries and smaller data for more efficient performance. However, in case of complex queries, Oracle switches to parallel execution and uses flags to determine the degree of parallelism.

Similarly, Apache has developed potent parallelism techniques around big data analytics. They have developed an open-source framework called Apache Hadoop which connects to all of their modules. The heart of their system lies in the MapReduce model which they have developed to function in conjunction with their Hadoop ecosystem. The MapReduce model of Hadoop is built up on the classic MapReduce model with some added fault tolerance and error handling mechanism. Additionally, Hadoop has also built their own file system known as HDFS to improve the robustness of their system. HDFS is built up on the Google File System with additional tweaks to achieve a high degree of parallelism and robustness.

Lastly, Apache has also built Spark to support streaming data. As streaming data is most likely associated with real-time applications and machine learning or clustering algorithms, Spark also has its own Machine Learning library called the MLlib. The Spark Architecture also follows a master / slave system and has efficient error handling mechanisms for parallel processing.

6. ACKNOWLEDGEMENTS

The author, Abhishek Manoj Sharma, would like to thank Dr. Robert K. Chun for his support and guidance during this project.

7. REFERENCES

[1] Li Dongming, et. al "Research on private cloud platform of seed tracing based on Hadoop parallel computing," 2015 4th Int. Conf. on Comp. Sci., Harbin, 2015, pp. 134-137.

[2] Herzing, "Importance of Data Warehousing?". [Online]. Available: herzing.edu/blog/importance-data-warehousing

[3] Jack B. Denni. 1974. A computer architecture for highly parallel signal processing. In Proceedings of the 1974 annual ACM conference - Volume 2 (ACM '74), Vol. 2. ACM, New York, NY, USA, 402-409. DOI=10.1145/1408800.1408808

[4] F. Dehne, "Parallel Real-Time OLAP on Multi-core Processors," 2012 12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, 2012, pp. 588-594. doi: 10.1109/CCGrid.2012.19

[5] Oracle, "Parallel Execution in Data Warehouses". [Online]. Available: docs.oracle.com/cd/E11882_01/server.112/e25554/px.htm

[6] M.Colgan, "In-Memory Parallel Execution in Oracle Database 11gR2". [Online]. Available: https://blogs.oracle.com/datawarehousing/in-memory-parallel-execution-in-oracle-database-11gr2

[7] S. Goil et. al, "A parallel scalable infrastructure for OLAP and data mining," Database Engineering and Applications, 1999. IDEAS '99. International Symposium Proceedings, Montreal, Que., 1999, 178-186. doi: 10.1109/1999.787266

[8] T. H. Sardar, A. R. Faizabadi and Z. Ansari, "An evaluation of MapReduce framework in cluster analysis," 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), 2017, pp. 110-114. doi: 10.1109/ICICICT1.2017.8342543

[9] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li, and Ying Li. 2009. Hadoop high availability through metadata replication. In Proceedings of the first int. workshop on Cloud data management (CloudDB '09). ACM, New York, NY, USA, 37-44. DOI=10.1145/1651263.1651271

[10] Todor Ivanov. 2018. Exploratory Analysis of Spark Structured Streaming in ACM/SPEC Int. Conf. on Performance Engineering (ICPE '18). ACM, New York, NY, USA, 141-146. DOI: 10.1145/3185768.3186360.