A Comparison of MapReduce and Parallel Database Management Systems

Alan McClean
School of Computing
Dublin Institute of Technology
Kevin Street, Dublin, Ireland
Email: d10123501@mydit.ie

Raquel. C. Conceição
Instituto de Biofísica e Engenharia Biomédica
Faculdade de Ciências
Universidade de Lisboa
Portugal
Email: raquelcruzconceicao@gmail.com

Martin O'Halloran
Electrical and Electronic Engineering
National University of Ireland Galway
Ireland
Email: martin.ohalloran@nuigalway.ie

***Abstract*—Businesses have come to rely on their data warehouse as a key component in their Information Technology infrastructure. The costs of the architecture to support these environments are significant. Therefore, choosing the wrong architecture can be a very costly decision. However, considerable confusion exists in relation to MapReduce and Parallel Database Management Systems (DBMS). In the past, MapReduce has been presented as a replacement for the Parallel Database Management Systems, as an additional tool that works alongside the Parallel DBMS, but also as an inferior tool by others. This paper will consider the broader themes of the paradigms rather than the specific implementations of MapReduce and Parallel DBMS. It will discuss MapReduce and Parallel Database Management Systems as competing and complimentary paradigms. The aim of this paper is to provide a high-level comparison between MapReduce and Parallel DBMS, providing a selection of criteria which can be used to choose between MapReduce and Parallel DBMS for a particular enterprise application.**

***Keywords*-MapReduce; Parallel Database Management Systems**

I. INTRODUCTION

In 2008, the world's servers processed 9.57 zettabytes of information. For every worker, there is approximately three terabytes of information created every year [1]. These high volumes of data has significant potential to improve understanding, leading to scientific breakthroughs and business process improvements. Examples of these improvements include personalized genome sequencing, extracting real-time trends from business analytics and social network based recommendations. However, the amount of data collected is now outpacing improvements in data storage technology [2]. The traditional data analysis approach is to load the data into a database and the use of a query language to perform the analysis. MapReduce has been presented as an alternative method, with implementations in leading IT companies such as Google and Facebook.

MapReduce is a method for processing large volumes of distributed data, allowing the use of shared nothing clusters. In distributed architectures, a shared nothing cluster is one where the nodes of the cluster share neither a common disk space, a common CPU or common memory. The program flow of MapReduce can be subdivided into a number of stages. Firstly, the data is parsed and some computation is completed. These tasks are referred to as the "Map tasks". Next, data is repartitioned across all nodes of the cluster. Finally, a second set of tasks are executed in parallel by each node on the partition of data it receives. These tasks are called the "Reduce" tasks, taking the results of the map tasks and combining them together [3].

Conversely, Parallel DBMS can be defined as Database Management systems that run over multiple nodes. They support standard relational tables, normally partitioned over multiple nodes and the use of the Structured Query Language (SQL) [3].

Several existing studies have compared MapReduce and Parallel DBMS. Some presented MapReduce as more flexible and capable of producing better performance [4]. Others have presented the opposite opinion [5]. Finally, some suggestions that the two approaches can be used together [6]. These inconsistent opinions have resulted in considerable confusion, making it difficult to determine the most appropriate technology for a particular application.

The aim of this paper is to provide a broad comparison of the two technologies, suggesting the most appropriate solution for a particular enterprise application. The structure of the paper is as follows: Section II will introduce MapReduce; Section III will describe Parallel DBMS; Sections IV and V will examine them as competing and complementary paradigms, while the results and conclusions will be discussed in Sections VI and VII.

II. MAPREDUCE

MapReduce stems from work first completed by Google, which was introduced by Dean *et al.* in 2004 [4]. The creation of the MapReduce library came about because "people at Google implemented hundreds of special-purpose computation that process large amounts of raw data" [4]. These included crawled documents and web requests logs. The computations themselves were relatively simple, but the input data was large and out of necessity the computation was distributed across many machines. This caused a number of issues, which kept repeating across different computations. These included how to:

 ISBN: 978-1-61208-246-2

- parallelize the effort;
- distribute the data;
- handle machine or node failures.

The MapReduce library was created as an abstraction. It allowed the developer to express the simple computation while hiding the details of parallelization, fault-tolerance, data distribution and load-balancing in the library.

MapReduce tasks run on top of Distributed File Systems (DFS). These systems allow files to be spread over multiple nodes which are connected by a network. DFS can also be extended to support fault tolerance. If one of more nodes in the DFS fails, then all data is still available. MapReduce allows the user to focus on implementing the logic required to solve their particular problem.

To do this, the user must write a program which integrates with the MapReduce library. The code must support two interfaces, "Map" and "Reduce". The library runs the Map code on each node in the cluster without communication with other nodes. The results of the map tasks are stored on the DFS. The reduce tasks are then run over the resulting data, to combine the outcome of the Map Results. When performing complex MapReduce algorithms, the general approach is to add additional MapReduce cycles, rather than trying to solve all items in a single parse [3].

Google created their own version of a DFS, commonly refered to as the Google File System (GFS). The exact implementation of MapReduce is only available to Google. However, the paradigm has been implemented in a number of places. Amongst these are:

- Apache Hadoop, support by Yahoo;
- Elastic MapReduce, at Amazon;
- Neptune by Ask.com;
- Dryad by Microsoft.

Of these, Hadoop has attracted the most interest, both commercially and academically. This is partly because of the open source nature of the project and partly because of the strong support and commitment from Yahoo. Apache Hadoop has its own file system (Hadoop Distributed File System (HDFS)). HDFS is highly fault tolerant and is designed to run on low cost components. Hadoop also allows streaming access to the data. HDFS is portable form one platform to another .

The basic MapReduce approach has been extended in a number of ways, particularly to improve performance. The following are examples where the paradigm has been extended:

- Map Join Reduce: This extends the MapReduce paradigm to improve performance by adding join tasks. While chaining of MapReduce jobs can be used to produce a multiple step join, Map Join Reduce completes this in a single step. This can lead to performance enhancements [8].
- MARS Accelerating MapReduce with Graphics Processors: In his study, Fang *et al.* [9] looked at extending MapReduce in a different way. The authors propose the use Graphics Processors rather than standard processors. These processors are faster but introduce a number of issues specifically regarding synchronization;
- Hadoop DB: This uses MapReduce as communication layer. Each node in the cluster has its own DBMS instance. Hadoop DB queries are written using SQL. The queries are translated into MapReduce code using extensions of existing functionality. The main body of the work is performed by the individual database nodes [3]

MapReduce is widely used by Google. In their original paper, Dean *et al.* suggested a number of problems to which it can be deployed [4]. These included:

- Machine learning problems;
- Clustering issues for Google News;
- Extracting data to produce reports on popular queries;
- Web Page property extraction;
- Processing of satellite imagery data;
- Statistical Machine Translation;
- Large scale graph computation.

Outside of Google, the MapReduce paradigm has also been widely used. The following are some sample commercial applications:

- Hive is an open source data warehousing solution built on top of Hadoop at Facebook. It is used for ad-hoc queries and for reporting into dashboards. The system is also used in machine learning algorithms. The system is used by both novices and experts [10].
- Nokia has deployed Hadoop cluster, based on Cloudera's commercial cluster into production. This is considered the companies enterprise wide information core.
- Four Square is a social network that allows its users to check in their location. Other users can then exchange information such as restaurant reviews and travel tips. Four Square use Amazon's Elastic MapReduce to perform a wide range on analytics on their data.

In conclusion, MapReduce is a paradigm first developed by Google. Its purpose is to abstract the developer from the complexities of handling large volumes of data in the computation. It is implemented as a library, which encapsulates the handling of parallelization, fault tolerance and data distribution. The approach has been widely used, particularly the open source version Hadoop. The alternative to MapReduce, Parallel DBMS, is described in the next section.

III. PARALLEL DBMS

Parallel DBMS were developed to improve the performance of database systems. As processor performances improvements outstripped disk throughput, critics predicted that I/O bottleneck would be a major problem. Despite this

 ISBN: 978-1-61208-246-2

a number of vendors, such as Teradata and Tandem, have brought successful products to market [11].

The main reason for the success of Parallel DBMS is the success of relational DBMS systems, now the dominant type of database system. Parallel DBMS have come into existence in a number of different project simultaneously. The DIRECT DBMS project was devised by David J. DeWitt in 1979. He extended this to the Gamma project. Simultaneously, Teradata was devised from research at the California Institute of Technology (Caltech) and from the discussions of Citibank's advanced technology group [12].

Within Parallel DBMS, there is a number of different types of architectures:

- Share memory: multiple processors share memory space and access to disks;
- Share Disk: multiple processor share access the same disk space, but each has its own memory;
- Share Nothing: each node has its own memory and disk space.

The key point is that shared-nothing architectures move only questions and answers between the nodes, while the other two architectures will move data through an interconnection network. The main advantage of the shared-nothing multiple processors is that they can be scaled up to hundreds and potentially thousands of processors that do not interfere with one another.

Parallel DBMS are a stable environment. There are a number of companies which rely on their parallel DBMS as a cornerstone of their business. Due to business confidentiality reasons, it is difficult to get exact figures on the volumes of data. However, in 2008, Teradata announced that it had five "petabyte power players". These included "an online auction company with 5.0 petabytes of data in their Teradata environment; a retailer with 2.5 petabytes; two large financial service institutions with 1.5 and 1.4 petabytes respectively, and a manufacturer with a one petabyte data warehouse environment" [12].

IV. Competing Paradigms

This section will consider applications where MapReduce and Parallel DBMS technologies have competed.

A. Large Data Volumes

Both MapReduce and Parallel DBMS provide a means to process large volumes of data. As the volume of data captured continues to rise, questions have been asked as to whether the parallel DBMS paradigm can scale to meet demands. "There are no published deployments of parallel database with nodes numbering into the thousands" [3]. As more nodes are added into the parallel DBMS environment, the chance of a node failure increases. Parallel DBMS do not handle node failure. MapReduce has been designed to run on thousands of nodes and is inherently fault tolerant. It has been presented as a viable alternative to parallel DBMS. This has been disputed by parallel DBMS experts. It is therefore necessary to establish an agreed comparison mechanism.

B. Analytics

Both MapReduce and Parallel DBMS can be used to produce analytics results from big data. Parallel DBMS uses SQL as the retrieval method, while MapReduce uses programming languages. In many data mining and data clustering applications, the algorithm is complex and requires multiple passes over the data. The output from one subprocess is the input to the next. It is difficult to develop these algorithms in SQL. The aggregation in SQL is not able to process these multiple step data flows. Performing these tasks in many steps reduces the performance benefits gained from parallel DBMS. For these complex analytic algorithms, MapReduce provides a good alternative [6].

V. Complementary Paradigms

In this section, the use of parallel DBMS and MapReduce as complementary paradigms is considered. One problem associated with parallel DBMS is the time taken to load large volumes of data. If this data is repeatedly queried then the load times impact needs to be averaged over each query. MapReduce, having a simple data structure does not suffer from the same load time issue. Therefore, MapReduce can be useful for one-time queries while parallel DBMS can be useful for repeated queries. This property of MapReduce makes it ideal for transforming data prior to load into the Parallel DBMS.

Another area where parallel DMBS can struggle is complex analytics. It can be quite difficult to express some queries in SQL. MapReduce allows the user to build complex computations on the data, without the limitation of the SQL language. In conclusion, parallel DBMSs are excellent at efficient querying of large data sets. Conversely, MapReduce is much slower by comparison, but is significantly better at complex analytics and ETL tasks.

There is a number of different approaches which have been taken to integrating parallel DBMS and MapReduce. These efforts have focused on using Hadoop implementation of MapReduce. Two of these approaches will be considered here:

- Loading Hadoop data in the Parallel DBMS;
- Accessing DBMS data from Hadoop;

Each of these will be further described in the following subsections.

A. Accessing DBMS data from Hadoop

This approach was first developed by Cloudera in the form of the *DBInputFormat* class. The basic approach is for each Map node to run the same SQL statement. The statement is modified to include order by, limits and offsets on the data. This ensures that each node receives a unique set of data. The main issue with this approach is that the performance

 ISBN: 978-1-61208-246-2

is often poor, as the same SQL is essentially run for every Map node. This approach has been extended by Teradata. In their version, the SQL statement is run once, and the output stored in a temporary table. This table is partitioned by the number of MapReduce nodes. When the MapReduce node requires the data, the query provides the data based on the appropriate partition [13].

B. Accessing Hadoop data from within the Parallel DBMS

DBMS provide a mechanism to integrate to external data using User Defined Function (UDF). Once complete the workings of the UDF are transparent to the database user. This allows the database user to write SQL query calls as follows:

*INSERT INTO Tab1 SELECT * FROM TABLE(udfLoadHadoop('mydfsfile.txt')) AS T1;*

This query will load the data from the Hadoop file, called *mydfsfile.txt*, into the database table *Tab1*. The UDF function is called "udfLoadHadoop". This function will take the Hadoop file as a parameter and integrate the data using the Hadoop NameNode metadata. The NameNode identifies which nodes in the Hadoop file system contain the required data. The UDF performs calculations based on the file size and the number of parallel nodes to determine which data belongs to each node. It then requests that data using the NameNode [13]

VI. RESULTS

This section will provide a comparison of MapReduce and parallel DBMS, across a broad range of criteria:

- Data Volume:
  - Parallel DBMS - Has been used for data volumes in the order of Petabytes;
  - MapReduce - Has been used for data volumes in the order of Petabytes;
- Cost:
  - Parallel DBMS - Enterprise level toolset. This is an expensive investment;
  - MapReduce - This is open source based solution. The investment is considered inexpensive;
- Fault Tolerance:
  - Parallel DBMS - Transaction Level, cannot survive node failure;
  - MapReduce - Fault tolerant, designed to survive multiple node failures;
- Users:
  - Parallel DBMS - Can be used by multiple user types, from Business Users using reporting tools, through SQL novices and expert users;
  - MapReduce - Requires programming skills to work with. Smaller pool of individuals capable of performing these tasks;
- Data Types:
  - Parallel DBMS - Supports structured data. Data has to be transformed into rows and columns;
  - MapReduce - Supports both structured and unstructured data. Data can be operated on in native format;
- Hardware:
  - Parallel DBMS - Homogenous, all nodes in the installation must be the same;
  - MapReduce - Heterogeneous, the nodes in the installation can be different. This allows for the use of commodity PCs;
- Maturity:
  - Parallel DBMS - Parallel DBMS have a long history of successful installation;
  - MapReduce - MapReduce is a relatively new technology. The source is continuously under development with new features being added;

VII. CONCLUSIONS AND FUTURE WORK

Based on this data, for the following types of problems, the authors suggest that MapReduce is used over parallel DBMS:

- Unstructured data: If the primary data source is unstructured then the cost of transforming it and loading it into a parallel DBMS is prohibitive. Based on this, MapReduce would be a good candidate;
- Cost: If cost is the main driver for the organization, then MapReduce is the better candidate. Parallel DBMS systems are considered enterprise level tools, but this comes at a high cost;
- User skill level: If the organization has an available pool of high skilled developers then MapReduce is a good option. In addition, if the organization is one in which control of the data is important then MapReduce is also the better candidate.

Conversely, for the following types of problems, the author suggests that parallel DBMS should be chosen above MapReduce:

- Structured data: If the data is structured and will continue to be so for the foreseeable future, then parallel DBMS would be a good fit;
- Enterprise Level Support: If enterprise level support is important to the organization, then the parallel DBMS vendors would be the preferred option. Although there are companies that offer this (for example Cloudera), companies in the parallel DBMS have been providing this support for a longer time;

 ISBN: 978-1-61208-246-2

- User: If the user base is not technical, or the data is not the key focus of the business, then parallel DBMS are the better choice.

Finally, many tests have compared versions of MapReduce and parallel DBMS focus on single queries. This provides clear results for performance. However, in the author's experience, in production environments multiple queries are often run concurrently. In addition to this, queries will include different types of problems. Some will be aggregating data; some will be joining data; others will be performing complex analytics. While this data is being queried, it is common for more data to be written to the system simultaneously. To provide a thorough comparison, the author believes it is necessary to test how the systems perform under these circumstances. To do this it would be necessary to establish a set of requirements. A test bench would then be created which would allow for these requirements to be run repeatedly.

REFERENCES

[1] R. B. J.E. Short and C. Baru, "How Much Information 2010: Report on Enterprise Server Information," http://hmi.ucsd.edu/pdf/HMI_2010_EnterpriseReport_Jan_2011.pdf, Jan. 2011, [Online; accessed 27-Nov.-2012].

[2] P. Ranganathan and J. Chang, "(Re)Designing Data-Centric Data Centers," *Micro, IEEE*, vol. 32, no. 1, pp. 66 –70, Jan.-Feb. 2012.

[3] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 922–933, Aug. 2009, [Online; accessed 27-Nov.-2012].

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://0-doi.acm.org.ditlib.dit.ie/10.1145/1327452.1327492

[5] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 35th SIGMOD international conference on Management of data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 165–178. [Online]. Available: http://0-doi.acm.org.ditlib.dit.ie/10.1145/1559845.1559865

[6] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and parallel DBMSs: friends or foes?" *Commun. ACM*, vol. 53, no. 1, pp. 64–71, Jan. 2010. [Online]. Available: http://0-doi.acm.org.ditlib.dit.ie/10.1145/1629175.1629197

[7] Apache, "Apache HDFS," http://hadoop.apache.org/common/docs/current/hdfs_design.html, 2012, [Online; accessed 27-Nov.-2012].

[8] D. Jiang, A. Tung, and G. Chen, "Map-join-reduce: Toward scalable and efficient data analysis on large clusters," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, no. 9, pp. 1299 –1311, Sept. 2011.

[9] W. Fang, B. He, Q. Luo, and N. Govindaraju, "Mars: Accelerating mapreduce with graphics processors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 608 –620, april 2011.

[10] 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," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1626–1629, Aug. 2009, [Online; accessed 27-Nov.-2012].

[11] D. DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems," *Commun. ACM*, vol. 35, no. 6, pp. 85–98, Jun. 1992, [Online; accessed 27-Nov.-2012].

[12] M. O'Sullivan, "Teradata," http://www.teradata.com/newsrelease.aspx?id=7243, April 2012, [Online; accessed 27-Nov.-2012].

[13] Y. Xu, P. Kostamaa, and L. Gao, "Integrating hadoop and parallel dbms," in *Proceedings of the 2010 international conference on Management of data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 969–974, [Online; accessed 27-Nov.-2012].

 ISBN: 978-1-61208-246-2