UNIVERSITY OF BIRMINGHAM

Parallelised Data Processing

An investigation into the development of a tool for processing queries across a cluster of nodes.

Oliver Little

2011802

D.A. B.Sc Computer Science with Digital Technology Partnership (PwC)

Supervisor: Vincent Rahli

School of Computer Science

College of Engineering and Physical Sciences

April 2023

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

1.2 Prior Work

Distributed Data Processing has existed conceptually since as early as the 1970s. A key paper by Philip Enslow Jr. from this period [8] sets out characteristics across three 'dimensions' of decentralisation - hardware, control and database. Enslow argued that these dimensions defined a distributed system, while also acknowledging that the technology of the period was not equipped to fulfil the goals he laid out.

Research into solutions for distributed data processing has generally resulted in two kinds of solutions [16]:

- **Batch processing:** where data is gathered, processed and output all at the same time. This includes solutions like MapReduce [6] and Spark [17]. Batch processing works best for data that can be considered 'complete' at some stage.
- **Stream processing:** where data is processed and output as it arrives. This includes solutions like Apache Flink [5], Storm [15], and Spark Streaming [3]. Stream processing works best for data that is being constantly generated, and needs to be analysed as it arrives.

MapReduce [6], a framework introduced by Google in the mid 2000s, could be considered the breakthrough framework for performing massively scalable, parallelised data processing. This framework later became one of the core modules for the Apache Hadoop suite of tools. It provided a simple API, where developers could describe a job as a *map* and a *reduce* step, and the framework would handle the specifics of managing the distributed system.

While MapReduce was Google's offering, other large technology companies had similar solutions, including Microsoft, who created DryadLINQ in 2009 [9]. However, due to the massive success of MapReduce, Microsoft discontinued DryadLINQ in 2011.

MapReduce was not without flaws, and many papers were published in the years following its initial release which performed performance benchmarks, and analysed its strengths and weaknesses [11]. Crucially, MapReduce appears to particularly struggle with iterative algorithms, like the PageRank algorithm used by Google's own search engine. A number of popular extensions to MapReduce were introduced to improve the performance on iterative algorithms, like Twister [7] and HaLoop [4] both in 2010.

MapReduce's popularity also resulted in a number of tools being created to improve its usability and accessibility. Hive [14] is one such tool, which features a SQL-like language called HiveQL to allow users to write declarative programs that compiled into MapReduce jobs. Pig Latin [13] is similar, and features a mixed declarative and imperative language style that again compiles down into MapReduce jobs.

Further tools in the wider areas of the field were introduced around 2010, including another project by Google named Pregel [12], specialised for performing distributed data processing on large-scale graphs.

In 2010, the first paper on Spark [19] was released. Spark aims to improve upon MapReduce's weaknesses, by storing data in memory, and providing fault tolerance by tracking the 'lineage' of data. This means for any set of data, Spark knows how the data was constructed from another persistent, fault tolerant data source, and can use that to reconstruct any lost data in the event of failure. This in-memory storage, known as a resilient distributed dataset (RDD) [18] allows Spark to improve on MapReduce's performance for iterative jobs, whilst also allowing it to quickly perform ad-hoc queries. Effectively, Spark is strong at performing long batch jobs, as well as short interactive queries. This is something that I would like my solution to feature, as users of the framework will need to design long-running scripts to run on large amounts of data, as well as run ad-hoc queries to perform investigation.

Spark quickly grew in popularity, with a number of extensions being added to improve its usability, including a SQL-style engine with a query optimiser [2], as well as an engine to modify Spark to support stream processing [3]. A second paper released in 2016 [17] stated that Spark was in use in thousands of organisations, with the largest deployment running an 8,000 node cluster holding 100PB of data. One area where Spark struggles is with grouped data, as performing grouped operations requires shuffling the data between all nodes. I aim to improve upon this in my solution through the design of the system as a whole.

More recent research indicates that the future of the field is moving away from batch processing, and towards stream processing for data that is constantly being generated. A 2015 paper by Google [1] argues that the volumes of data, the fact that datasets can no longer ever be considered 'complete', along with demands for improved insight into the data means that streaming 'dataflow' models are the way forward. Google publicly stated in their 2014 'Google I/O' Keynote [10] that they were phasing out MapReduce in their internal systems. The data I will be using is not being received at this constant rate, and as such designing for a streaming solution is not required in this case.

1.3 Project Aims

Chapter 2

Design

F = Functional

NF = Non-Functional

M = Must

S = Should

C = Could

F / NF	M / S / C	Requirement Description
Data Processing		
F	M	The system must allow users to filter datasets according to custom criteria.
F	M	The system must allow users to join two datasets together according to custom criteria.
F	M	The system must allow users to group portions of datasets together by unique keys.
F	M	The system must allow users to apply custom functions to each row of a dataset.
F	M	The system must allow users to apply a reducer function to a dataset.
NF	S	The complexities of the system should be hidden from the user; the operation of the system should be identical whether the user is running the code locally or over a cluster.
Cluster		
F	M	The user should be able to index specific columns of the data to allow these rows to be accessed more quickly by the cluster.
F	S	The cluster should be able to handle node failures by restarting them and requesting new work.
NF	S	The cluster should feature some form of load balancing to ensure that nodes are not idling if there is still work to be performed.
Main Nodes		
F	M	The main node must delegate work to the cluster nodes to perform the calculation efficiently.
F	M	The main node must collect the results from the cluster nodes to produce the final output for the user.

F	M	The main node must handle cluster node failures to ensure that the calculation completes successfully anyway.
Automated ETL Workflows		
F	S	The user should be able to define workflows, defined as the below requirements in this section *(Automated ETL)*, using some kind of configuration file.
F	S	The system should receive files sent from the user, ingest them into the system automatically, and start a calculation when the files are sent.
F	S	The user should start calculations manually, schedule them for a specific time, or schedule them at a regular interval.
F	S	The system should output the results of a calculation to a user-defined location.
F	S	The system should raise alerts when an unrecoverable failure occurs during a calculation.
F	C	The system could send alerts via email to the user when an unrecoverable failure occurs during a calculation.
F	C	The system could monitor a specific folder location for changes, then initiate a calculation when files in that location are changed.
Dashboard		
F	S	The system should have a dashboard to display basic information about user-created workflows *(from Automated ETL)*, including source and destination files, and previous executions.
F	C	The system could allow the user to start or schedule workflows from the dashboard.

Chapter 3

Implementation

Chapter 4

Testing

Chapter 5

Evaluation

Bibliography

[1] Tyler Akidau et al. "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing". In: (2015). URL: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43864.pdf.

[2] Michael Armbrust et al. "Spark sql: Relational data processing in spark". In: *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 2015, pp. 1383–1394. DOI: 10.1145/2723372.2742797. URL: https://doi.org/10.1145/2723372.2742797.

[3] Michael Armbrust et al. "Structured streaming: A declarative api for real-time applications in apache spark". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 601–613. DOI: 10.1145/3183713.3190664. URL: https://doi.org/10.1145/3183713.3190664.

[4] Yingyi Bu et al. "HaLoop: Efficient iterative data processing on large clusters". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 285–296. DOI: 10.14778/1920841.1920881. URL: https://doi.org/10.14778/1920841.1920881.

[5] Paris Carbone et al. "Apache flink: Stream and batch processing in a single engine". In: *The Bulletin of the Technical Committee on Data Engineering* 38.4 (2015).

[6] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113. DOI: 10.1145/1327452.1327492. URL: https://doi.org/10.1145/1327452.1327492.

[7] Jaliya Ekanayake et al. "Twister: a runtime for iterative mapreduce". In: *Proceedings of the 19th ACM international symposium on high performance distributed computing*. 2010, pp. 810–818. DOI: 10.1145/1851476.1851593. URL: https://doi.org/10.1145/1851476.1851593.

[8] Philip Harrison Enslow. "What is a" distributed" data processing system?" In: *Computer* 11.1 (1978), pp. 13–21. DOI: 10.1109/c-m.1978.217901. URL: https://doi.org/10.1109/c-m.1978.217901.

[9] Yuan Yu Michael Isard Dennis Fetterly et al. "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language". In: *Proc. LSDS-IR* 8 (2009).

[10] *Google I/O Keynote*. 2014. URL: https://youtu.be/biSpvXBGpE0?t=7668.

[11] Kyong-Ha Lee et al. "Parallel data processing with MapReduce: a survey". In: *AcM sIGMoD record* 40.4 (2012), pp. 11–20. DOI: 10.1145/2094114.2094118. URL: https://doi.org/10.1145/2094114.2094118.

[12] Grzegorz Malewicz et al. "Pregel: a system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 135–146. DOI: 10.1145/1807167.1807184. URL: https://doi.org/10.1145/1807167.1807184.

[13] Christopher Olston et al. “Pig latin: a not-so-foreign language for data processing”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008, pp. 1099–1110. DOI: 10.1145/1376616.1376726. URL: https://doi.org/10.1145/1376616.1376726.

[14] Ashish Thusoo et al. “Hive-a petabyte scale data warehouse using hadoop”. In: *2010 IEEE 26th international conference on data engineering (ICDE 2010)*. IEEE. 2010, pp. 996–1005.

[15] Ankit Toshniwal et al. “Storm @twitter”. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 147–156. DOI: 10.1145/2588555.2595641. URL: https://doi.org/10.1145/2588555.2595641.

[16] Ibrar Yaqoob et al. “Big data: From beginning to future”. In: *International Journal of Information Management* 36.6 (2016), pp. 1231–1247. DOI: 10.1016/j.ijinfomgt.2016.07.009. URL: https://doi.org/10.1016/j.ijinfomgt.2016.07.009.

[17] Matei Zaharia et al. “Apache spark: a unified engine for big data processing”. In: *Communications of the ACM* 59.11 (2016), pp. 56–65. DOI: 10.1145/2934664. URL: https://doi.org/10.1145/2934664.

[18] Matei Zaharia et al. “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing”. In: *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012, pp. 15–28.

[19] Matei Zaharia et al. “Spark: Cluster computing with working sets”. In: *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. 2010. URL: https://www.usenix.org/event/hotcloud10/tech/full_papers/Zaharia.pdf.