ISSN 2063-5346

MAPREDUCE MODEL FOR MEMORY AWARE OPTIMIZATION MA-OHMR USING BIG DATA WITH APACHE FLINK

Vaishali Sontakke [1], Dr. Chandrakala B M [2]

Article History: Received: 10.05.2023 Revised: 29.05.2023 Accepted: 09.06.2023

Abstract

A big data set consists of a large collection of data that has a wide variety of data characteristics, has a high growth rate and has many complex characteristics. An infrastructure capable of managing large volumes of real-time data is needed to process complex data, which is unstructured. The computation of big data will therefore require the use of MapReduce, which simplifies the process. In Hadoop Distributed File System (HDFS), MapReduce is used to compute data sets. MapReduce algorithms are adaptable to various types of transformation. It's possible to review different architectures of the MapReduce method with Apache Flink. This final project utilizes unstructured text data for data management. Implementing MapReduce on Linux operating systems and designing HDFS applications. In Apache Flink, the MapReduce program is used to count words. The results of this study show that Flink MapReduce is faster at performing computation than Hadoop MapReduce by 38%.

Keywords: *Big Data, Hadoop Apache, MapReduce, Apache Flink.*

[1] Research scholar, Dayananda Sagar College of Engineering, kumaraswamy layout, Bengaluru, India

[2] Associate Professor, Department of Information Science & Engineering, Dayananda Sagar College of Engineering, Kumaraswamy layout, Bengaluru, India

DOI:10.48047/ecb/2023.12.9.14

Introduction

There is rapid development in technology and a corresponding increase in services. An emerging service is internet technology, which is growing rapidly. An Internet Protocol address allows all electronic devices with an IP address to communicate with each other. All users have access to software services, whether they pay for them or not. Data storage consumption increases with the development of complex software. A large amount of data is consequently contained in an internet application. The term "Big Data" refers to data that is large in scale. It is a large and diverse collection of data that can be unstructured and is growing rapidly. It is possible for big data to run slowly if the computer used to process the data does not meet the necessary standards.t in-depth information is easily obtained and can help make better decisions [1]. In the case of big data, Hadoop is the solution. The Apache license enables Hadoop to be used for applications that run on large amounts of data. Hadoop's origins can be traced back to a 2003 paper by scientists from Google, Jeffrey Dean and Sanjay Ghemawat on Google MapReduce and Google File System (GFS). Hadoop is named after Doug Cutting's son's toy elephant, which is also the name of his computer program. A distributed file system called Hadoop Distributed File System (HDFS) provides storage and computation in a distributed manner. A programming model is used to distribute clusters of computers/nodes in Hadoop. It provides scalable and easy data processing solutions through the use of MapReduce in the background of Hadoop [2]. It is possible to process structured, semistructured, and unstructured data through data processing. In order to optimize the process of sending data from map tasks to reduce tasks, low latency, high-throughput memory caches are being replaced with distributed low latency, high-throughput memory caches. The computation process uses Apache Flink functions to meet these requirements. In-memory batch and stream processing can be implemented with Apache Flink, an open source platform for Mapreduce programs. In Flink, there are two main APIs: DataSet API, which processes finite data sets (often referred to as batch processing), and DataStream API, which processes infinite data streams (often referred to as stream processing). The research will process unstructured text data. Mapreduce batch processing is implemented using Linux's HDFS and Mapreduce batch processing. This research is highlighting further work upon following objectives

a) Developing an HDFS-based application for processing Big Data. b) Improving the Mapreduce function based on HDFS. c) Identifying the response time and resource utilization of the Mapreduce method based on memory, processor, and disc usage. d) Comparing the efficiency of Apache Flink compared with Hadoop Mapreduce method.

Identifying the problem the problem in making this research paper proposing the slow access to big data and the lengthy computation process of Hadoop Mapreduce method. The data sets are large and there are a variety of variations to manage. Rapidly growing data (up to date).In big data, both structured and unstructured data are collected in large quantities. Business strategies and decision-making can be improved by analysing big data. It was introduced in the 2000s by industry analyst Doug Laney, who introduced Big Data, which consists of three important components: Volume, which is a large collection of data that can't be processed by a single computer, and Availability, which is the ability of an organization to use large amounts of data. As a result of the length of time required to process this data, it cannot be processed traditionally. Velocity, with large data, the collection of data will flow quickly and has never

happened in previous technologies that must be handled in a timely manner. The data are available in a variety of formats, including structured data, numeric data in traditional databases, and unstructured data.

2. RELATED WORK

In Apache Hadoop, several computers are connected using the Java programming language so that they can work together and synchronize in storing and processing data. YARN is a framework that facilitates the negotiation of multiple resources. Hadoop Distributed File System (HDFS), Mapreduce, Hadoop Common are four components that makeup Apache Hadoop. Mapreduce jobs and resource management are managed by YARN, a framework that handles job scheduling [3]. Figure 1 shows HDFS Architecture.

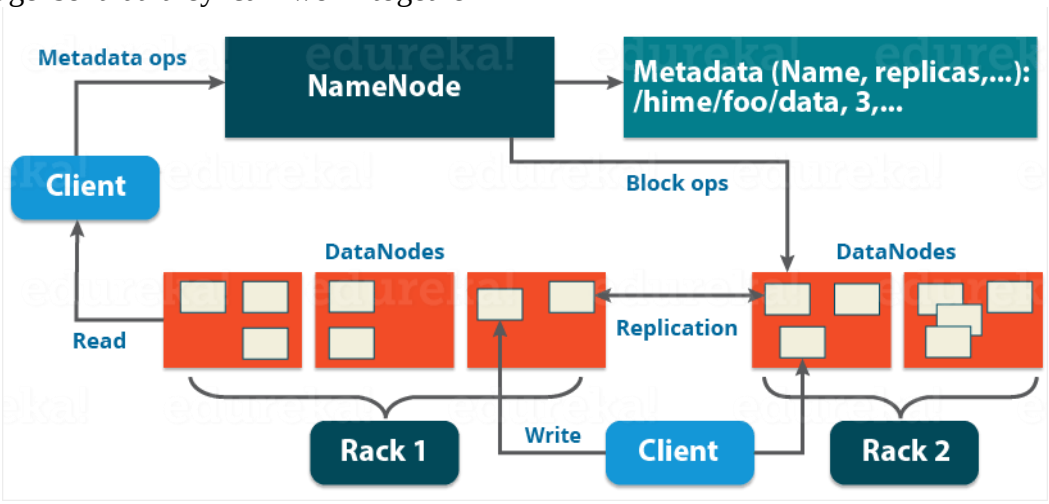

Figure 1 HDFS Architecture

In addition to the GNU/Linux operating system, Name Nodes have software that implements the Name Node system. Generally, HDFS systems have a Name Node that serves as a master server for metadata and file system operations. It manages the file system namespace, organizes client access to files, and performs file system operations such as renaming, closing, and opening files. GNU/Linux and Data Node software are installed on the Data Node, which is a commodity device. There will be several Data Nodes in a cluster. HDFS system data is stored on this node. File system operations are performed by Data Nodes based on client requests. As directed by the Name Node, perform operations such as creating, deleting, and replicating blocks. HDFS files contain blocks that store user data. Individual data nodes or segments will be used to store files in the file system. A block is a segment of a file. As a result, HDFS can only read or write a minimal amount of data. As needed, HDFS configuration can be changed to increase block size from 64MB to 128MB [4].

This programming model is used to process large data sets efficiently. There are three stages in MapReduce, namely the map stage, the shuffle stage, and the reduce stage. A single stage, the reduce stage, combines the shuffle and reduce stages. By combining the Map function and Reduce function, developers can create a MapReduce application. In step one, the map stage, input data is processed. Generally, HDFS files are used as input, and then the files are converted into tuples, which are pairs of keys and values. During the reduce stage, the map input data is processed and the shuffle and reduce stages are performed, and the new data set is saved back to HDFS. As an example of how to map and reduce work, we have the following illustration. A Mapper Class tokenizes input, maps it and sorts it by

key-value pairs. The Reducer Class uses the output of the Mapper Class as input and then groups them according to the same value key. Figure 2 shows the Mapper & Reducer

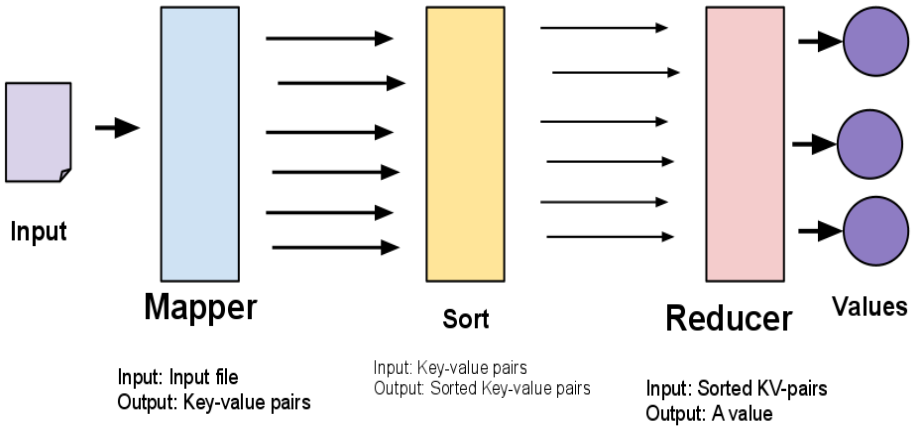

Figure 2 Mapper & Reducer

A big data computation can be performed using Apache Flink. There are many storage systems from which Flink can read or write data. For distributed computation over data streams, Flink consists of a dataflow streaming engine that distributes data, communicates data, and provides fault tolerance. In addition to native overlay iteration and memory management, Flink also supports batch processing on top of its streaming engine. Hadoop's MapReduce process can be performed by Apache Flink. A Hadoop Mapper implements the Mapreduce method on Apache Flink, equivalent to a FlatMap function. Group Reduce is equivalent to Hadoop Reducer. A number of file systems can be used to run Apache Flink, including HDFS [6][7].

3. PROPOSED METHODOLOGY

In this area of research Flink's architecture can be summarized as follows:

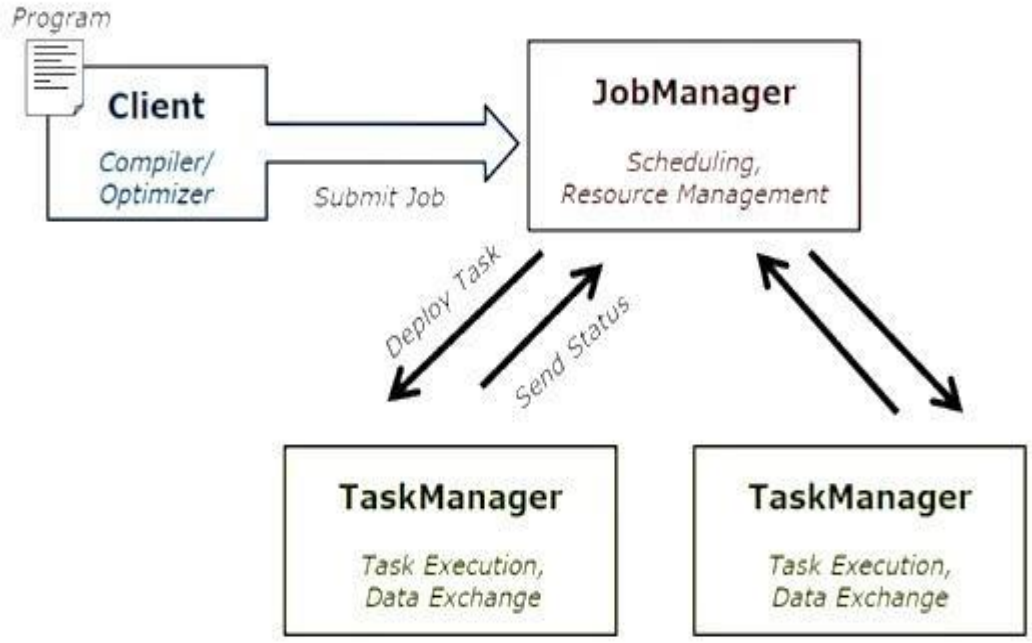

Figure 3 Apache Flink Architecture

MapReduce tasks are pushed to clusters via Job Tracker, which allocates tasks to clusters with the highest priority data, thereby maximizing their performance. After a client submits a job to a Job Tracker, it talks to the Name Node to determine where the data is located, then it determines which node is closest to the data, and it submits the task to that node. The Job Tracker will get a signal when the Task Tracker completes a task and assign it another. The Name Node machine is used by default [8],[9], Figure 3 shows the Apache Flink Architecture. Task Managers are called workers because they run tasks from the data stream and buffer and swap it. There are a set of slots in each Task Tracker, and each slot accepts a task. The JobTracker first looks for an empty slot on the server holding the DataNode that contains the data, if not on that server, then on a machine in the same rack that holds the Data Node. The actual job is run in a separate JVM process, so if the job crashes, Task Tracker won't be affected. Task Tracker keeps track of the output and returns values of the task while it runs. The Job Tracker is notified after the task has been completed.

In this research, HDFS installation and configuration will be used to design the system. Then Apache Flink is installed and configured. In order to run Mapreduce on HDFS, the Hadoop environment must be set up on Apache Flink after the configuration and installation processes are complete. HDFS stores local data in text files that are fed into Hadoop from local data sources after Hadoop is installed. There are several types of data, each with different file size. A small amount of data is tested first, then a larger amount of data is tested gradually. HDFS will be used to process the data using the Hadoop Mapreduce programme. A two-stage testing process will be followed. There are two specific ways to implement Mapreduce, namely the disk-based technique on Hadoop and the in-memory batch processing with HDFS using the Mapreduce method on Flink.

4. PERFORMANCE EVALUATION

The following test scenarios will be performed in this study: In the tests, response time and computer performance were measured. Data processing speed and resources used are measured with this test. The following section contains two files with different sizes, ranging from small to large, with .txt and .csv extensions: 1. 1.6 GB for Facebook-names-unique 2. 5.2 GB Facebook-names-original size

4.1 Processing time

Here for test the data, five different data files has taken. Using Hadoop Mapreduce & Flink Mapreduce on HDFS comparison carried out on these different data set. An average of five tests has taken, makespan time in table1 and table 2 for Hadoop Mapreduce for various size of data files are 1.6, 2.5, 3.4, 4.3,5.2 are considered and existing model observations are 276,284,289,298,304.. Here improvisation observed in terms of processing time using Flink Mapreduce.

Response Time using Flink Mapreduce	
Data files in GB	Processing time in sec
1.6	276
2.5	284
3.4	289
4.3	298
5.2	304
Average	285

Response Time using Flink Mapreduce		
Testing	Processing time (seconds)	
	1.6 GB file	2.5 GB file
1	178	264
2	179	264

3	177	265
4	179	266
5	180	267
Average	179	265

Resource Utilization

According to Table 2, testing on files measuring 1.6 GB takes about 179.2 seconds. It takes about 265,713 longer for files measuring 2.5 GB. In comparison to the test results in Table 2, shows that Flink Mapreduce on HDFS can be run very quickly due to the use of in-memory processing. The output produced by Flink Mapreduce is also more stable. The standard deviation of Flink Mapreduce is smaller than that of Hadoop MapReduce, as can be seen from the results of the standard deviation test. *The purpose of this test is to determine how much memory is needed in processing data to determine which one uses more resources.* For resource usage, we will observe several components. HDFS and Yarn are used in Hadoop MapReduce. It is important to know that HDFS has three active components, namely: Data node, Name node, and Secondary Name node. There will be two active components in Yarn as the Hadoop Mapreduce engine: Resource manager and node manager. HDFS and the Task manager and Job Manager Components on Flink are all that are required for MapReduce on Flink. The results of testing Hadoop Mapreduce on various files of different sizes are shown in Table 3. The use of memory resources in the first file is relatively low at 29.66%, while the processor consumes many resources. The test results indicate that the processor works harder and uses less memory than in the first file experiment. Memory usage increased in the second experiment. It is an increase of 0.66%. A batch Flink Mapreduce test was then run in memory. The memory conditions will change when Apache Flink is run. Memory and processor resources are used more heavily. The memory and processor graphs do not change much for every process that runs. Comparing the read and write processes performed by Hadoop Mapreduce and the read and write processes on the disc, the read and write process on the disc has decreased. When processing very large files, memory and processor will reach their maximum. A small file may still be able to use memory and processor if the heap is still able to provide resources.

Test Results for Flink Mapreduce Performance From Table 3, below are the results of testing Hadoop Mapreduce performance on various file sizes. The first file uses relatively few memory resources around 29.66%, and the processor consumes a large amount of resources. According to the test results, the processor worked more than in the first file experiment, and memory resources were used less. A rise in memory usage was observed in the second experiment. It is an increase of 0.66%. Flink Mapreduce was then tested in-memory in a batch mode. The memory conditions will change when Apache Flink is run. The processor and memory are the most heavily used resources. There is not much change in the processor graphs and memory graphs after every process runs. On the other hand, when viewed from the read/write process on the disc, it has decreased in comparison with Hadoop Mapreduce. When processing very large files, memory and processor usage will peak. In this case, the use of memory and processor is not too high if the file is small and the memory heap is still able to provide resources. In the table below, you will find the test results. The results in Table 4 above show the performance of the HDFS-based Flink MapReduce batch in-memory on the files tested. Flink Mapreduce does not require high processor resources for small files with heap memory capacity. In the event that the processed file exceeds the heap memory configured, the percentage of processor resources will be increased [7].

Analysis

Analyzing the time response to the Mapreduce process performed by HDFS is the first step in the analysis. Two platforms are supported by the Mapreduce method. After implementing the first on Hadoop, the second on Apache Flink with an addition of some transformation functions. A word count program is used to implement the MapReduce method. A standalone or local node is used for testing. It has been proven that Hadoop Mapreduce is capable of performing disk-based computations after testing. In addition, Apache Flink provides in-memory batch processing. MapReduce on Flink performs computations more quickly than MapReduce on Hadoop [8]. There is no way to determine with certainty the performance and resources required when testing Hadoop Mapreduce. As a result, the graph obtained is always fluctuating. As a result of the stability of the graph, test results obtained with Flink Mapreduce are more accurate. Using files in sizes larger than 1GB, you can see a noticeable difference in speed. The comparison graph can be found here. Based on Figure 4, the Mapreduce method can be implemented on Flink to speed up computations. There is a 37.18% increase in computation time with the Mapreduce method. This is because Mapreduce on Flink has a different method in processing files. In Tables 3 and 4 it is evident the read and writes process that occurs on the disc when performing Mapreduce on Hadoop. The read and write process on Hadoop Mapreduce is greater than Flink Mapreduce. This causes Hadoop Mapreduce to be slower. For the read and write process on Flink Mapreduce, that is by sending micro-batch data into memory gradually and continuously. A micro batch of data is sent in the same way as a streaming data stream. Due to this, Flink Mapreduce can perform read and write tasks simultaneously, saving both time and effort.

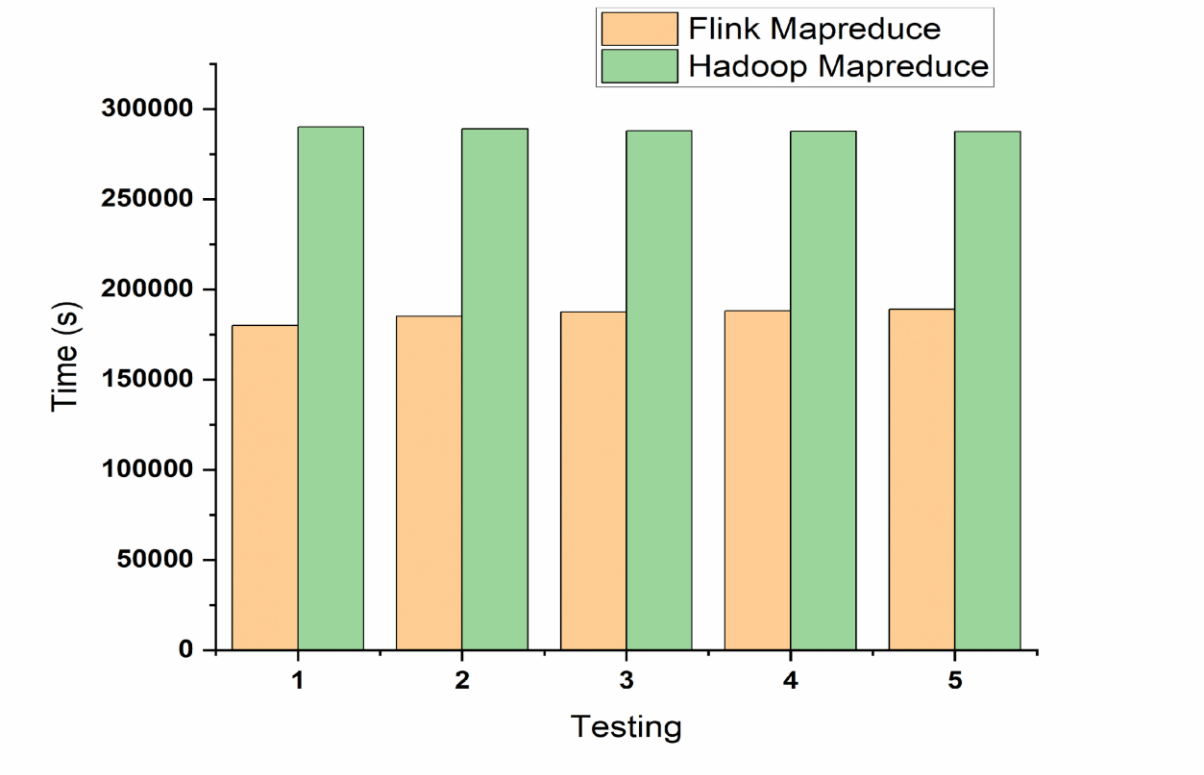

Figure 4. Response time analysis graph

MA-OHMR (Memory Aware optimized Hadoop MapReduce)

This other research work proposes MA-OHMR (Memory Aware optimized Hadoop MapReduce), MA-OHMR is developed considering memory as the constraint and prioritizes memory allocation and revocation in mapping, shuffling and reducing, this further enhances the job of Mapping and reducing. Optimal memory management and I/o operation are carried out to use the resource inefficiently manner. The model utilizes the GMM (Global Memory Management) to avoid garbage collection and MA-OHMR is optimized on the makespan front to reduce the same. MA-OHMR is evaluated considering two datasets i.e. simple workload of Wikipedia dataset and complex workload of sensor dataset considering makespan and cost as an evaluation parameter. Upon evaluation of datasets varying the different data sizes, MA-HMR is proven to be efficient and cost-effective.

1) In the proposed MapReduce model the job on one computing node is executed as a thread, and all the job threads can share the memory dynamically at the runtime.
2) The proposed model adopts sequential disk access due to the Thread-based technique.
3) Balance the data production of CPU and data consumption of disk I/Os the proposed model utilizes the multi-buffer technique, which implements the non-blocking I/O.
4) The proposed model considers hybrid merge-sort instead of simply external sorting from disks, in order to minimize the size of spilled intermediate data on disks that help in realizing the in-memory
5) The proposed model avoids re-reading them from disks before transferring them to remote reduce tasks by caching the final merged files output by Map tasks in memory and solving the problem of full Garbage Collection in the Java virtual machine
6) For effective coordination of the above strategy, these data buffers are managed by a global memory controller and an I/O scheduler on each node.

Conclusion

It can be concluded from the test results that Mapreduce can be implemented in a variety of ways. The MapReduce method can be transformed to work like Hadoop with Apache Flink's transformation function. A key feature of Apache Flink is that it allows Mapreduce to send near-real-time micro-batch data to memory in order to speed up computing. As opposed to Hadoop MapReduce, Apache Flink MapReduce is twice as fast as MapReduce on Apache Flink. The server computer must have a large hard drive and memory to achieve maximum results. It can be concluded that Flink can speed up computations by implementing the Mapreduce method. There is an increase in computation time of 37.18% when using the Mapreduce method. The reason for this is that MapReduce on Flink processes files differently. As shown in Tables 3 and 4, Hadoop performs Mapreduce on the disc by reading and writing data. Hadoop MapReduce is faster at reading and writing than Flink MapReduce. The result is slower Hadoop MapReduce. In Flink MapReduce, micro batches of data are continuously and gradually read and written into memory. The micro-batch data is transmitted in the same way as streaming data. As a result, Flink MapReduce can perform the read and write processes simultaneously, thereby saving processing time.

Data availability statement

The data used to support the findings of this study are included in the article

Funding statement

The authors declare that no funding was received for Research and Publication. It was performed as a part of the Employment of Institutions.

References:

[1] R. S. K. V. and D. N. P. Kavya, "Big Data Processing by harnessing Hadoop - MapReduce for Optimising Analytical Workloads," IEEE,2014.

[2] S. Zhang, J. Han, Z. Liu, K. Wang and S. Feng, "Accelerating MapReduce with Distributed Memory Cache," 15th International Conference on Parallel and Distributed Systems, 2009.

[3] M. Junghanns, A. Petermann, N. Teichmann, K. Gómez and E. Rahm, "Analysing Extended Property Graphs with Apache Flink," ISBN,2016.

[4] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015.

[5] X. Wu, "A MapReduce Optimisation Method on Hadoop Cluster," International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information, 2015.

[6] S. Baltagi, Overview of Apache Flink: Next-Gen Big Data, Chicago: Chicago Apache Flink Meetup, 2015.

[7] V. Markl, "Breaking the Chains: On Declarative Data Analysis and Data Independence in the Big Data Era," Proceedings of the VLDB Endowment, 2014.

[8] K. Tzoumas, "Apache Flink," 2015. [Online]. Available: https://www.slideshare.net/KostasTzoumas. [Accessed 4 December 2017].

Table 3 Hadoop Mapreduce Performance Testing

No.	1.6 GB file				2.4 GB file			
	CPU	Disc kB/s		Memory	CPU	Disc kB/s		Memory
		Read	Write			Read	Write	
1	91 %	10343	836	56 %	89 %	11088	748	63 %
2	90 %	10292	362	55 %	89 %	11324	694	63%
3	90 %	10532	818	62 %	89 %	11449	663	64 %
4	90%	10526	1174	62 %	89 %	11090	1009	64 %
5	90%	10465	1217	63 %	89 %	11188	897	64 %
Avg	90%	10431	881	60 %	89%	11228	802	64 %

Table 4 Flink Mapreduce Performance Testing								
No.	1.6 GB file				2.4 GB file			
	CPU	Disc kB/s		Memory	CPU	Disc kB/s		Memory
		Read	Write			Read	Write	
1	88 %	12764	5074	27 %	88 %	10966	3565	30 %
2	92%	10715	4198	30 %	83%	10861	3646	30 %
3	91%	10585	4002	30 %	92%	11837	3506	30 %
4	92%	9595	3511	29 %	90 %	11025	3495	30 %
5	91 %	10857	3723	29 %	90%	11241	3518	30 %
Avg	91%	10903	4101	29 %	89%	11186	3546	30 %