WEP Warwick Evans Publishing

International Journal of Computer Science and Information Technology

ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 4, Number 3, Year 2024
DOI: https://doi.org/10.62051/ijcsit.v4n3.39
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Analysis of Parallel Optimisation Strategies Based on MapReduce Models

Shuhang Wu

School of Internet, Anhui University, Hefei, Anhui230000, China

ABSTRACT

The aim of this paper is to provide an in-depth analysis of parallel analysis strategies for MapReduce models, and to explore how to improve the overall performance by optimising task allocation and scheduling, improving data locality and increasing node utilisation. The research methodology includes an analysis and overview of existing MapReduce frameworks and proposes a series of improvement strategies. These strategies improve the utilisation of computing resources by adjusting the granularity of task division, optimising data slicing and distribution, and improving task scheduling algorithms. The results show that by reasonably optimising the parallel analysis strategy of the MapReduce model, its performance in large-scale dataset processing can be significantly improved, especially in resource-constrained distributed environments. Ultimately, this paper concludes that although the MapReduce model has been used in more mature applications, there is still much room for optimising its parallel strategy when facing larger scale and complex data processing tasks. In the future, further research should be devoted to finer-grained task scheduling, dynamic resource allocation, and more efficient fault-tolerance mechanisms to continuously improve the parallel processing capability of the MapReduce model.

KEYWORDS

MapReduce; Parallel computing; Big data; Spark; Performance tuning; Task scheduling

1. INTRODUCTION

In the field of data processing, traditional stand-alone processing is often difficult to cope with the rapid growth of data volume and complex computational demands. With the popularity of the Internet and the acceleration of the digitalisation process, the volume of data has expanded dramatically, and traditional databases and computing systems often face performance bottlenecks when dealing with large-scale data.

These bottlenecks include: Storage constraints: the limited storage capacity of a single computer makes it difficult to meet the demand for large-scale data storage. Computing power: A single machine has limited computing power, and complex data processing tasks may not be completed in a reasonable amount of time. Scalability problem: traditional data processing models are difficult to scale horizontally, i.e., performance improvement is not obvious when computing resources are increased. Therefore, the MapReduce model was proposed by Google in 2004 to solve the corresponding problems.

The analysis of parallel optimisation strategies based on MapReduce model is of great research significance in improving the efficiency of big data processing, optimising resource utilisation, and enhancing system reliability and fault tolerance. Through in-depth study of these optimisation strategies, improvement schemes can be proposed for the performance bottlenecks of the MapReduce model, which can significantly increase the data processing speed, ensure the reasonable allocation

and utilisation of resources, and reduce the task execution time and data transmission overhead. At the same time, these optimisation strategies can also enhance the stability and fault tolerance of the system, ensuring stable operation in the face of various abnormal situations. In addition, the research results can provide theoretical and technical support for the development of cloud computing, edge computing, Internet of Things and other emerging technologies, promote industry application innovation, provide efficient data processing solutions for various industries such as finance, healthcare, manufacturing, logistics, etc., and help enterprises realise data-driven innovation and transformation, and promote socio-economic development.

2. MAPREDUCE BASIC CONCEPTS

2.1. Overview of the Model

MapReduce is a programming model for processing and generating large-scale data sets. It enables computations to be parallelised and distributed across multiple computing nodes by decomposing the computational task into two main steps: map and reduce. The framework was proposed by Google [1] in 2004 and has excelled in processing large-scale datasets.

2.2. Modelling Workflow

(1) The workflow of MapReduce starts with data partitioning, where the input data is divided into multiple data blocks, each of which is processed by a Map task. The Map task reads the data blocks and applies the Map function to generate intermediate key-value pairs, which are then cached and ready to be sent to the Reduce task.

(2) After the Map task is completed, the Shuffle and Sort phases begin. The Shuffle phase is responsible for grouping the intermediate key-value pairs according to the keys to ensure that all values of the same key are sent to the same Reduce task. The Sort phase sorts these key-value pairs to facilitate efficient processing of the Reduce task.

(3) The Reduce task receives the key-value pairs passed in the Shuffle and Sort phases, and applies the Reduce function to aggregate each key and its associated values to generate the final output. These results are then written to the distributed file system for subsequent analysis and use.

2.3. Example Demonstration

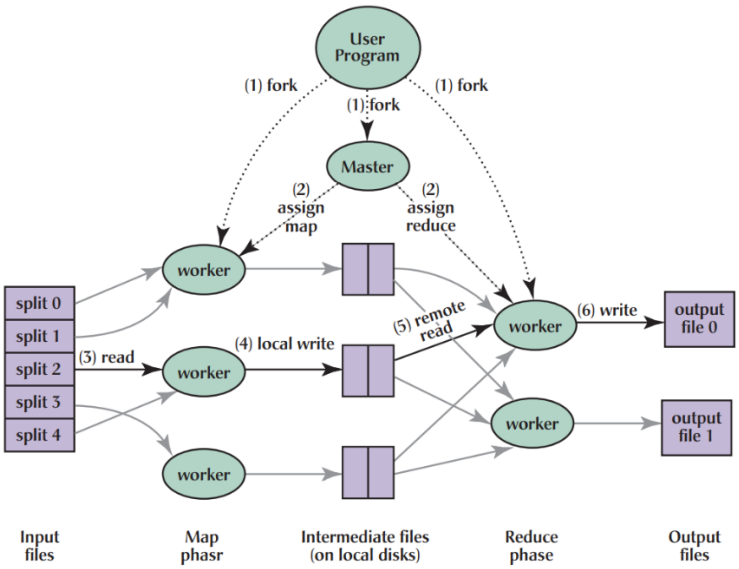

Figure 1. Execution overview.

If we have a large document and we want to count the number of times each word appears in the text, we can use the MapReduce framework to do so. First, the document is divided into smaller chunks, each of which serves as input to a Map task, which reads the text in the chunks and outputs each word and its number of occurrences (with an initial value of 1) as an intermediate result. Next, the Shuffle phase aggregates the same words and sorts them. The output of the Shuffle phase is a list of each word and all its corresponding intermediate values. Finally, the Reduce task receives the output of the Shuffle phase, accumulates the list of values for each word, calculates the total number of times each word appears in the entire document, and outputs the final result. In this way, we get the total number of occurrences of each word in the document.

3. MAPREDUCE PARALLEL STRATEGY OPTIMISATION ANALYSIS.

3.1. Task Scheduling and Resource Allocation

The Existing MapReduce implementations usually use a static task scheduling strategy, but uneven resource utilisation may occur during task execution, resulting in some nodes being overloaded or idle. By introducing a dynamic load balancing strategy, the allocation of tasks can be dynamically adjusted according to the real-time running condition of the tasks, thus improving the resource utilisation of the whole cluster.

In order to reduce the latency associated with data transmission, the task scheduler should try to assign tasks to the nodes where the data is located (data localisation scheduling). Further optimisation may include considering rack-aware scheduling, i.e., in cases where data localisation is not possible, tasks are preferentially assigned to nodes within the same rack.

3.2. Task Execution and Parallelism

The By adjusting the number of Map and Reduce tasks, it is possible to better adapt to different sizes of datasets and computational resources. For example, by adjusting the slice size of Map tasks or increasing the number of Reduce tasks, the problem of long task execution time or resource waste can be avoided. For frequently updated datasets, traditional MapReduce needs to recompute the full amount of data. By introducing incremental MapReduce, only new or updated data can be computed, greatly reducing the overhead of repeated computation.

3.3. I/O Performance Optimisation

Between the Map and Reduce phases in MapReduce there is usually a process of writing data to disk, which can lead to unnecessary I/O overhead. In some cases, it is possible to chain multiple Map and Reduce tasks to reduce the number of intermediate results written to disk and pass the data directly through memory. When executing similar tasks multiple times, performance can be further optimised by using a local cache to store the intermediate results, reducing the number of accesses to the distributed file system.

3.4. Fault Tolerance and Robustness

When a task fails, the MapReduce model typically re-executes the entire task. If the task can be decomposed and recovered at a fine-grained level, only the failed part can be re-executed, unnecessary computational overhead can be reduced. In long-running tasks, the intermediate state of the task is saved periodically, so that when the task fails, it can continue execution from the nearest checkpoint instead of starting from scratch, thus improving fault tolerance.

3.5. Data Compression and Serialisation Optimisation

MapReduce generates a large amount of I/O and network traffic during data transfers, which can significantly impact performance, especially when dealing with massive amounts of data. Use efficient compression algorithms (e.g., Snappy [2], LZO [3]) to reduce the amount of data transferred. Also, optimise the data serialisation and deserialisation process and use efficient serialisation frameworks (e.g. Avro, Thrift, Protocol Buffers) to reduce serialisation overhead.

3.6. Input Data Format Optimisation

The performance of MapReduce is closely related to the format of the input data. Unstructured or complex data formats increase processing time. Choosing the right data format (e.g., Parquet, ORC), which is usually columnar, can reduce I/O and speed up data reads.

4. MAPREDUCE COMBINED WITH SPARK

4.1. Introduction to Spark

Apache Spark [4] is an open source distributed computing framework developed by the AMPlab team at UC Berkeley. Similar to MapReduce, Spark is used to process large-scale data, but it is designed to compensate for MapReduce's performance shortcomings, especially in iterative computation and in-memory processing.

4.2. Key Features of Spark

(1) In-memory computation: Spark saves intermediate data in memory, which reduces the dependence on disk I/O and greatly improves computation speed, especially when multiple iterations of computation are required.

(2) Flexibility: Supports multiple modes of operation, including Batch Processing [5], Stream Processing [6], Interactive Querying [7], and Machine Learning [8].

(3) Rich API: Provides a clean and easy-to-use API that supports Java, Scala, Python, and R programming.

4.3. MapReduce and Spark Technology Convergence

4.3.1. Complementary task processing

MapReduce for batch tasks: Since MapReduce excels at handling one-off, large-scale data batch processing tasks, it remains ideal in environments where stability and fault tolerance are required.

Spark for iterative and real-time tasks: in scenarios that require iterative computation or real-time data processing, Spark is better suited due to its in-memory computing advantages.

4.3.2. Data sharing and interoperability

Hadoop Ecosystem Integration: Spark seamlessly integrates with the Hadoop ecosystem, reading data stored in HDFS and processing it. In this way, users can leverage their existing Hadoop infrastructure while enjoying the performance gains that Spark brings.

Hybrid Architecture: Some organisations using Big Data may choose to start with MapReduce for initial data cleansing and batch processing, and then use Spark for further real-time analysis and machine learning of the processed data.

4.3.3. Conversion and migration

Migration from MapReduce to Spark: Some systems with existing MapReduce tasks can be migrated to Spark to take advantage of Spark's faster computational power and rich operational capabilities to improve overall performance and efficiency.

5. FUTURE DEVELOPMENT TRENDS

5.1. Integration with Artificial Intelligence

In the future, MapReduce may be more integrated with artificial intelligence (AI) technologies. For example, integrating MapReduce with machine learning and deep learning frameworks to optimise the process of handling large-scale training data and improve the efficiency of AI model training.

5.2. Real-time Data Processing

MapReduce will face challenges as the demand for real-time data processing increases. Future research may focus on how to further extend MapReduce to support real-time stream processing tasks, which may require the introduction of new parallel programming models or deep integration with existing stream processing frameworks such as Apache Kafka and Apache Flink.

5.3. Resource Utilisation and Green Computing

Improving resource utilisation and achieving green computing will become an important research direction in the future. The research may focus on how to reduce the waste of computational resources through smarter resource scheduling and task allocation strategies, and at the same time reduce the energy consumption of MapReduce clusters.

5.4. Security and Privacy

With the increasing prominence of data security and privacy protection issues, future research in MapReduce will pay more attention to data encryption, access control and privacy protection mechanisms. Especially when dealing with sensitive data, how to ensure data security while ensuring computational efficiency will become a key research direction.

5.5. Deep Integration with Cloud Computing

The integration of MapReduce with cloud computing will continue to deepen, and future research will explore how to better deploy and optimise MapReduce applications in cloud environments. Serverless Computing and Auto-scaling are likely to be the focus.

6. CONLUSION

By reviewing and analysing various existing research papers on parallel analysis strategies for the MapReduce model, we provide an in-depth discussion on the current state of the model's application in large-scale data processing, the challenges it faces, and possible optimisation directions. In this paper, we review the main research results, covering a number of key areas such as task scheduling, load balancing, data distribution and locality optimisation, and fault-tolerance mechanisms. Through comparative analysis of different approaches, we identify the most effective parallel strategies in specific scenarios, and also reveal the shortcomings of current research.

Overall, although the MapReduce model performs well in processing large-scale data, its parallel analysis strategy still needs to be further optimised and improved with the rapid expansion of data

size and growing computational demands. This study finds that the performance of the MapReduce model can be significantly improved by combining finer-grained task partitioning, dynamic load balancing, improved data locality strategies, and more robust fault tolerance mechanisms. In addition, combining with the latest hardware development and distributed computing technology, future research should focus on developing more intelligent and adaptive scheduling algorithms to cope with changing computing environments and more complex data processing requirements.

REFERENCES

[1] Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. in Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04) (pp. 137-150).

[2] Abadi, D. J., Boncz, P. A., Harizopoulos, S., Idreos, S., & Madden, S. (2010). Column-stores vs. row-stores: How different are they really? In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (pp. 967-980). ACM.

[3] Oberhumer, M. (1995). LZO real-time data compression library. Retrieved from http://www.oberhumer.com/opensource/lzo/

[4] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. in Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12) (pp. 15-28). USENIX Association.

[5] Desler, G. A. (1964). A batch processing system. Proceedings of the AFIPS Spring Joint Computer Conference (pp. 115-121).

[6] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Schneider, S., Yadav, J., Kulkarni, S., Jackson, J., Dai, Y., Baldeschwieler, E., Bhagat, N. Mittal, S., & Ryaboy, D. (2014). Storm@twitter. in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (pp. 147-156). ACM.

[7] Agarwal, S., Milner, H., Kleiner, A., Talwadker, S., Kavulya, S., Mozafari, B., Madden, S., & Stoica, I. (2013). BlinkDB: Queries with bounded errors and bounded response times on very large data. in *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13) * (pp. 29-42). ACM.

[8] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 210-229.