Word frequency statistics based on MapReduce on serverless platforms

Yuheng He[1], Jin Qian[2], Juanjie Zhang[3], Renzhe Zhang[4,5]

[1]School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, China
[2]School of Mathematics and Statistics, Xuzhou University of Technology, Xuzhou, China
[3]School of Communications Engineering, Xidian University, Xi'an, China
[4]School of Computer Science, Chongqing University, Chongqing, China

[5]20220691@stu.cqu.edu.cn

Abstract. This paper investigates the application of serverless computing in conjunction with the MapReduce framework, particularly in machine learning (ML) tasks. The MapReduce programming model has been widely used to process large-scale datasets by simplifying parallel and distributed data processing. This study explores how the combination of these two technologies can provide more efficient and cost-effective ML solutions. Through a detailed analysis of serverless environments and the MapReduce framework, this paper shows how the combination can advance the fields of cloud computing and machine learning. The experimental part includes the implementation of Map-Reduce model on a serverless platform, exploring the impact of different parameter settings on performance and improving efficiency by optimizing the data processing flow. In addition, the paper analyzes the use of memory and CPU resources and derives the relationship between dataset size, memory consumption and processor configuration and execution time. Through these experiments and analyses, this paper provides an empirical basis and theoretical support for the optimization of cloud computing frameworks.

Keywords: MapReduce, Serverless Computing, Cloud Computing, Efficient and Cost-Effective

1. Introduction

As cloud computing technology continues to advance, people have witnessed an evolution from Infrastructure as a Service (IaaS), Platform as a Service (PaaS) to Software as a Service (SaaS). These stages of development provide flexible, scalable solutions for modern computing. However, while these models have made significant progress in simplifying the management and utilization of IT resources, they still do not fully address the issues of resource waste and management burden. Against this backdrop, serverless computing has emerged as an emerging model of cloud-native computing and has quickly become the focus of industry attention [1].

The core advantage of serverless computing is its "pay-as-you-go" model, which means that users only need to pay for the computing resources they actually use, thus significantly reducing the waste of idle resources. This model is particularly well suited for applications that do not need to run continuously, such as machine learning (ML) tasks, which tend to have sudden resource demands and different workflow phases. However, users of ML applications face the dual challenges of over-resourcing and explicit resource management, which are particularly prominent on traditional server computing platforms [2].

To address these challenges, in 2008, Google engineers proposed the MapReduce computing framework in order to solve the problem of processing massive datasets - the user generates a set of intermediate key/value pairs by specifying a map function, which processes the key/value pairs to generate a set of intermediate key/value pairs, and a reduce function, which function merges all intermediate values associated with the same intermediate key to achieve the purpose of processing data in chunks. In the ensuing years, other engineers have improved it many times and developed many new computing frameworks, and MapReduce has greatly simplified the complexity of data processing. Google and other organizations have successfully used MapReduce to process various types of data, including crawler documents, web request logs, etc., as well as to generate a variety of derived data [3].

Since the MapReduce framework is not applicable to acyclic data flow models, the Spark framework was created, which introduced Resilient Distributed Datasets (RDDs) that allow RDDs to be explicitly cached on different machines for reuse in multiple parallel operations [4]. In 2011, Verma A and his team members developed a method to break the barrier between Map and Reduce stages, improving operational efficiency while maintaining generality and ease of programming. It was found that the performance improvement mainly comes from dropping disk-intensive work and interleaving I/0 with computation [5]. In the same year, Nipype, an open-source Python-based software package, was released, which is similar to the MapReduce computational framework and is intended to be used for the efficient development of neuroimaging data and fast comparison of algorithms [6]. In 2013, Wang G's team proposed two new multitasking optimization techniques for the MapReduce framework, including a generalized grouping technique and a materialization technique, and designed a new optimization algorithm that significantly improved the operational efficiency of the framework [7]. After this, Z. Tari built an accountable MapReduce for detecting malignant nodes to solve the possible dishonest situation of MapReduce [8]. In 2015, Apache Hadoop based on MapReduce framework undeniably became the most accepted open source system for big data analytics, which is similar to the data processing framework Apache Flink, which also serves as an open source model for processing streams and batch processing, and which uses data stream processing as a real-time in the programming model and execution engines analytics, continuous streaming, and batch processing as a unified model. 2022, scholars such as Wenjun Qian summarized the problems of a series of computing frameworks including MapReduce in terms of privacy leakage as well as a variety of privacy-preserving techniques, which pushed the MapReduce framework for further improvement [9]. Recently, some researchers try to compare MapReduce and Apache Spark to determine their merits, shortcomings, and applicability for big data applications and make a conclusion [10].

Given the significant advantages of serverless computing in terms of resource efficiency and ease of management, as well as the effectiveness of MapReduce in dealing with large-scale distributed computation, this study aims to explore how the combination of the two can provide a more efficient and economical solution for ML applications. Through an in-depth analysis of serverless environments and the MapReduce framework, this paper will illustrate the significance of this combination in advancing the fields of cloud computing and machine learning.

In order to optimize the Map-Reduce framework and test its performance, the paper first constructed the basic Map and Reduce functions, and achieved the division of labor among multiple sets of Map-Reduce functions by adjusting the parameters. After that, the group divided the dataset into more than one thousand groups and tested its effect on the running time of the program by setting the Map-Reduce framework with different number of groups. Also, the group investigated the ratio of function communication time to execution time and monitored the memory consumption of individual Map-Reduce functions on the AliCloud platform. Finally, the group varied the dataset size and tested the effect of different memory allocations on execution time to produce experimental results.

2. Research Methodology

2.1. Framework Environment

The paper implement a MapReduce framework based on two services of AliCloud's Function Computing FC (an event-driven fully managed serverless service) and Object Storage OSS to perform the work of counting the frequency of words.

2.2. Data Presentation

In this study, more than one thousand English original readings were randomly adopted and converted into 1476 txt files with a total size of 438.802 MB, which were uploaded to Aliyun OSS database in the named formats of data0.txt, data1.txt, data2.txt ......data1476.txt waiting to be downloaded and called.

2.3. Map Design

2.3.1. Framing Components. In the Map phase, the framework designed in this paper is divided into four main parts:

1. receive parameters, provide error reporting code.
2. Validate based on the OSS2 library to obtain the raw data.
3. Process the raw data to get the filtered words.
4. Upload the processed data for use in the reduce phase.

2.3.2. Pseudocode Description. The description of the important parts of the map function is listed as follows:

Algorithm 1 Initialize the OSS Authentication Object

```
Authentication with AccessKey ID and AccessKey Secret
    Initialize the OSS authentication object
    Create a Bucket object based on the Endpoint and Bucket names
```

Algorithm 1 describes the authentication process to Aliyun OSS Database so as to transport data.

Algorithm 2 File Download

```
Record the timestamp of the start of the download
For each file ID in the range:
    Download files with from OSS Bucket to the local area
Record the download end time and calculate the total download time
Print the total time spent on the download task
```

Algorithm 2 describes the logic of downloading raw data in our frame.

Algorithm 3 Mapping Task Handling (Map1 & Map2)

```
define map1:
    Initialize 26 dictionaries
    For the first half of the files to be processed:
        Read the contents of the file
        Split content into raw words
        Iterate over the raw words
            Count raw words based on its initial letter
    Record upload start time
    Convert statistics into JSON format and upload to OSS
    Record the upload end time
```

```
        Calculate and print the total upload time
        Return
    define map2:
        Initialize 26 dictionaries
        For the *second* half of the files to be processed:
            The rest of the pseudocode is similar to *map1*
        ...
        Return
```

Algorithm 3 describes the procedure of filtering and counting the words, which is the core part of this program.

Algorithm 4 Multithreaded Execution

```
    Create and start thread for *map1*
    Create and start thread for *map2*
    Wait for all threads to finish
```

Algorithm 4 gives the simple description of the ways of multithread execution.

2.4. Reduce Design

2.4.1. Framing Components. In the Reduce phase, the framework of this paper is divided into four main parts:

1. receive parameters, provide error reporting code
2. based on the OSS2 library for validation, to obtain the data processed in the map phase
3. Integrate the output results of the map phase
4. upload the final result

2.4.2. Pseudocode Description. Considering that the logic of Reduce phase is similar to Map phase, the same pseudocode is omitted.

Algorithm 5 Mapping Task Processing (Reduce1 & Reduce2)

```
    define *reduce1*:
        for range *a* to *m*:
            Record download start time
            Download the corresponding files in *map1* and *map2*
            Merge files
            Record upload start time
            Write and upload the merged result
        Record and print the total communication time
        Return
    define *reduce2*:
        for range *n* to *z*:
            The rest of the pseudocode is similar to *reduce1*
        ...
        Return
```

Algorithm 5 shows how do the program combine the processed data together to get the final output.

3. Result

3.1. Effect of map and Reduce group size on function execution time

After the MapReduce function is run, the four original files to be processed are divided into 26 files named in alphabetical order and stored in the result folder defined in oss. During the reduce stage, all functions in the map are integrated into one file, which is the final result file. The final file contains all words in alphabetical order.

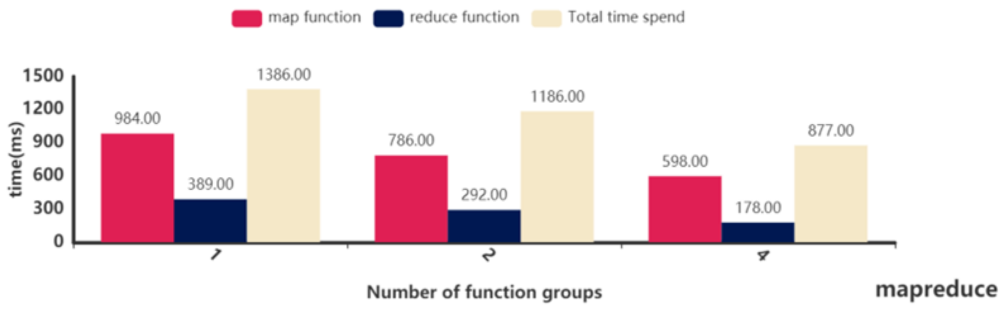

Figure 1. The time consumed by MapReduce function in the case of four data sets in division and the whole

The result of MapReduce function optimization can be reflected from the function execution time. In this experiment, the control variable method was used to explore the experimental results. The result is shown in Figure 1.

First change the number of map function groups individually. The result is that as the number of map functions increases, the time taken by the function decreases. The number of Map groups increased from 1 to 4, and the elapsed time decreased from 984ms to 598ms. Then, the number of reduce function groups is changed separately. As the number of reduce function groups increases, the time consumed by the functions gradually decreases, the number of reduce groups increases from 1 to 13, and the number of functions decreases from 389ms to 150ms. It is worth noting that as the number of groups increases, the time consumption interval between the two groups due to the increase of the number of groups also decreases. Finally, the map and reduce functions are combined to run, increasing the number of groups from 1,2,4. It can be considered that with the increase of the number of function groups, the function time consumption still presents a downward trend from 1386ms to 877ms. This shows that increasing the number of map and reduce functions can reduce the overall code run time.

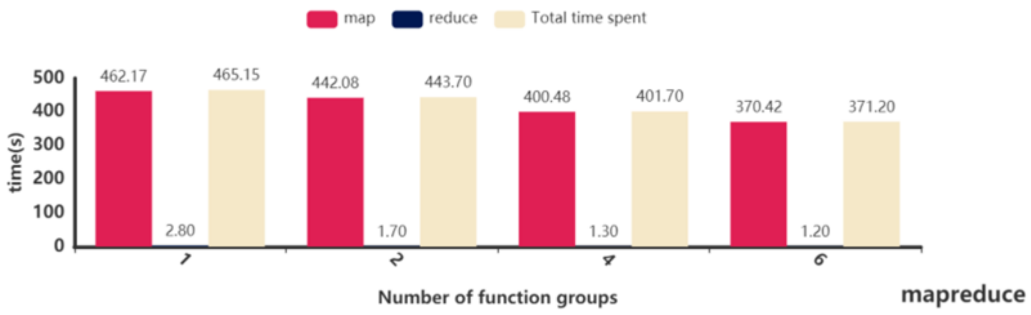

Figure 2. The time consumed by the MapReduce function in the case of 1477 data sets in the partial case and the overall case

In order to eliminate the influence of the data set, the second experiment was performed, and a larger data set was added, so that the map and reduce functions would collectively process about 1477 files. Repeat the above experiment, divide it into map function and reduce function, and finally merge the two functions and experiment again. As shown in Figure 2, the map and reduce function teams successfully conducted experiments to record the words of the extracted articles, and optimized the functions so that the code could eliminate factors such as Spaces and mismatches, and successfully integrated into a file. After the experiment, by changing the number of function groups, it is found that with the increase of the number of function groups, multiple workers can share the data and process it, which can reduce the time required by the function. At the same time, with the increase of the data set, the time of map function gradually occupies the total time of the function. Reducing the time of map function is conducive to reducing the overall time of the function faster.

However, there are still problems worth noting. As the number of map function groups and reduce function groups increases, the number of function threads increases, which increases the CPU load, and may fail to run the program completely.

3.2. Ratio of function execution time to communication time

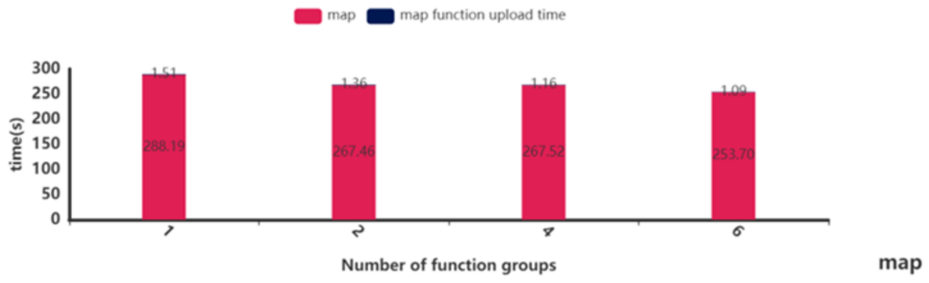

Figure 3. map function execution time and communication time

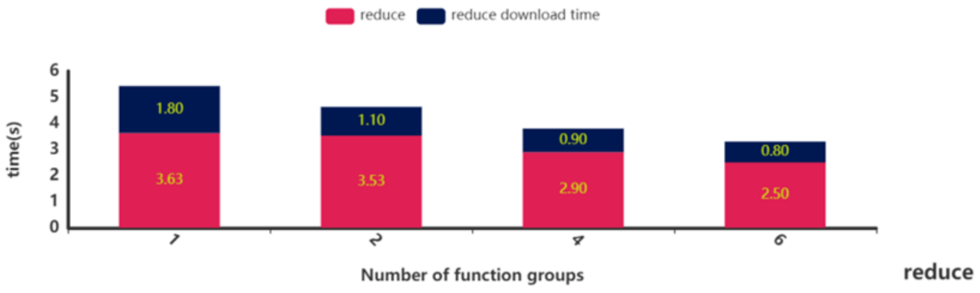

Figure 4. reduce function execution time and communication time

The change of communication time with the number of experimental groups in the total execution time is explored. Figure 3 shows the changes in the execution time and communication time of map function, and Figure 4 shows the changes in the execution time and communication time of reduce function. Therefore, according to the content in the figure, it can be concluded that after the execution of a set of Map functions, the time required to upload the output file is about 1.51s. The total execution time of the Map function is 288.19s. After a set of Reduce functions are executed, it takes about 1.242 seconds to download the output file of the Map phase. The total execution time of Reduce functions is 3.629 seconds. The calculated total communication time is approximately 2.752s and the total execution time is 291.819s. The results show that the proportion of communication time is about 0.943%.

The time required for the two sets of Maps to upload output files is about 1.36s. The total execution time of the Map function is 267.46s. After the two groups of Reduce functions are executed, it takes about 0.942 seconds to download the output file of the Map phase. The total execution time of Reduce function is 3.527s. After calculation, the total communication time of function is 2.302s, and the total execution time of model is 270.987s. The results show that the proportion of communication time is about 0.849%. The decline is also significant compared to the first group.

The time required for the four Map groups to upload output files is about 1.16 seconds. The total execution time of the Map function is 267.52 seconds. After the four groups of Reduce functions are executed, it takes about 0.896 seconds to download the output file of the Map phase. The total execution time of the Reduce function is 2.979 seconds. After calculation, the total communication time of the function is 2.056s, and the total execution time of the model is 270.499s. The results show that the proportion of communication time is about 0.76%. It takes about 1.08s for the six Map groups to upload output files. The total execution time of the Map function is 253.7 seconds.

After the six groups of Reduce functions are executed, it takes about 0.849 seconds to download the output file of the Map phase. The total execution time of Reduce functions is 2.198 seconds. After calculation, the total communication time of the function is 1.929s, and the total execution time of the model is 255.898s. The results show that the proportion of communication time is about 0.753%.

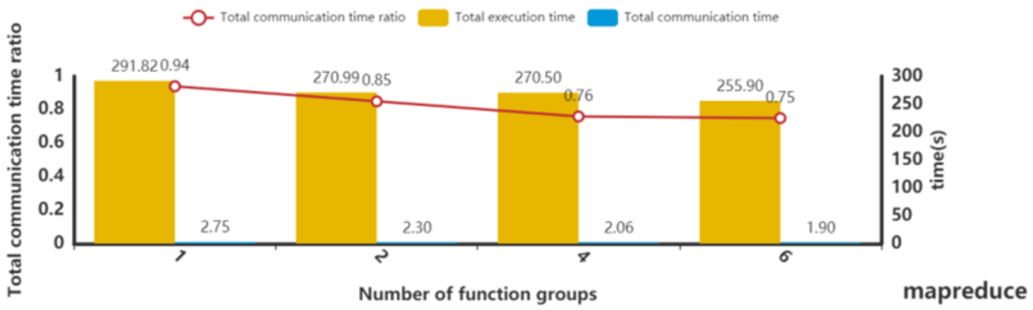

Figure 5. Total execution time and communication time and their proportion

The results are summarized and plotted. It can be intuitively observed from the line chart in Figure 5 that when python's multi-threading idea is used, with the increase of Map and Reduce functions, the communication time and total function execution time both show a downward trend. At the same time, the proportion of communication time in the total time is also gradually decreasing. Prove the feasibility and performance of the function. When the data set is enlarged, the performance of the model changes more obviously and the results are more reasonable.

3.3. Memory Consumption by Functions

3.3.1. Testing Environment Configuration. The following takes map function as the main test function for experimentation:

Performing MapReduce operations in a Serverless environment and measuring the memory consumption of the map function usually involves several steps. Since Serverless platforms (e.g. AWS Lambda, Google Cloud Functions, Azure Functions, etc.) usually provide monitoring and logging features, the paper can use these features to observe and analyse memory usage.

Platform selection: choose the AliCloud Serverless platform to run the experiment.

Function writing: Write the map function to ensure that the function can handle data sets of different sizes in order to observe changes in memory consumption.

Deploy the function: deploy the trigger function and observe the memory consumption, deploy the map function to the AliCloud platform. Ensure that the appropriate execution environment, memory limit and timeout settings are configured. Trigger the map function using different input dataset sizes and observe the memory consumption. View the memory usage of the function through the monitoring tools provided by the platform (e.g. AWS CloudWatch).

Record and analyse data: Record memory usage for each function execution and analyse the numbers to draw visual images to understand the relationship between memory consumption and dataset size.

Plotting: use the matplotlib library (a plotting library for Python) to plot graphs.

3.3.2. Result

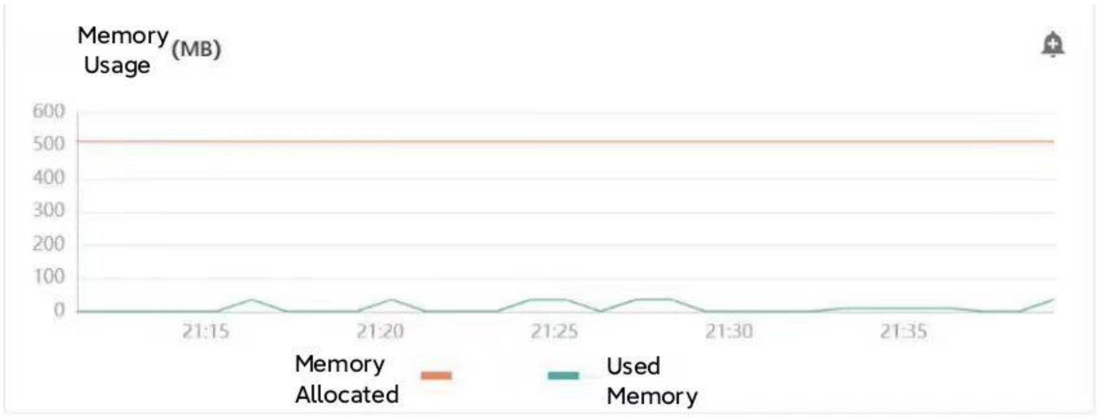

Figure 6. Memory Consumption

As shown in Figure 6, in terms of content usage, the map and reduce functions tend to use memory in a flat and regular way.

The paper conducted a series of experiments to test the memory usage of datasets of different sizes during the execution of the map function. The paper chose 3 datasets of different sizes (small, medium and large) and recorded the memory consumption of each dataset during the execution of the map function.

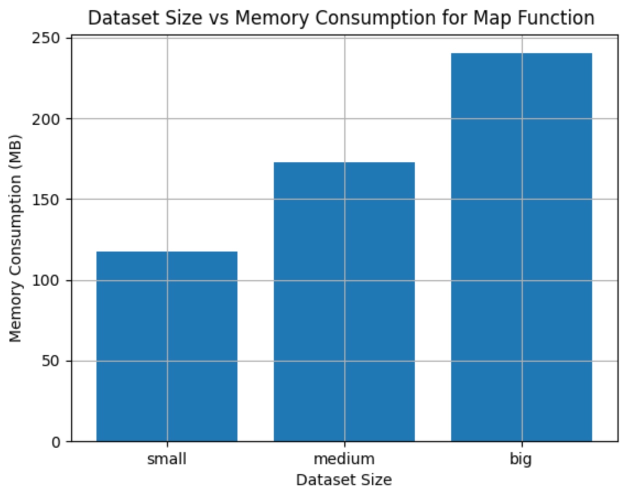

Figure 7. The Relationship with Data and Memory

The relationship between dataset size and memory consumption during the execution of the map function can be seen from Figure 7, as the dataset size increases, the memory consumption increases accordingly.

3.3.3. Conclusion. The paper conducted multiple experiments using a fixed dataset size to eliminate potential randomness while ensuring that all other conditions of the experimental environment remained constant. At the same time, attention was paid to the memory limitations of the serverless platform and ensured that the function ran within the set memory limits. It is finally concluded that the dataset size has a positive correlation with the memory consumption in the map function, i.e., as the dataset size increases, the memory consumption increases accordingly, while ensuring that other factors are not affected.

3.4. Effect of Memory and Processor Parameters on Execution Time

3.4.1. Theoretical Analysis. Memory size is an important factor in server performance. Too little memory and system processes will be blocked and applications will be slowed down or even unresponsive; too much memory can also be wasteful. Therefore, finding the right memory size is an important factor in balancing server performance and cost.

In general, the higher the number of cores of the processor and the higher the main frequency, the better the performance of the server. This is reflected in the serverless computing service provided by Aliyun, which is the parameter setting of vCPU. the higher the parameter setting of vCPU, the faster the speed of computing.

3.4.2. Experimental Content. The paper conducted MapReduce experiments in AliCloud and tested the effect of different memory allocations on execution time. The paper chose several different memory allocation levels (256MB, 512MB, 1024MB) and used a fixed size dataset. As shown in Table 1:

Table 1. The Setting of Testing Environment

vCPU	Execution time(ms)	memory (MB)
V0.25	12060	512
V0.25	12952	1024
V0.25	14224	2048

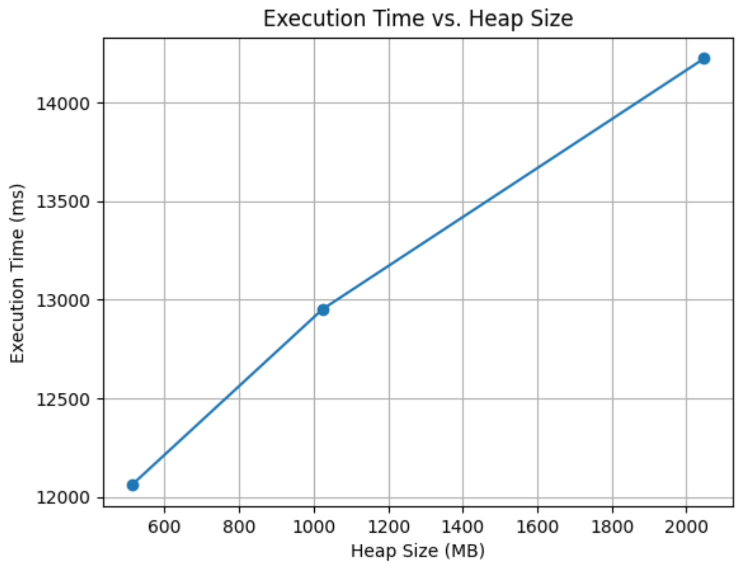

Figure 8. The Relationship with Data and Execution Time

The above figure 8 shows the relationship between the size of the dataset and the memory consumption during the execution of the map function. It can be seen from the figure that as the size of the dataset increases, the memory consumption also increases accordingly. In addition to memory, the paper also tested the effect of different CPU sizes on execution time. The paper chose several different CPU specifications (V0.25, V0.35, V0.5, V0.75) and used a fixed size dataset. The following experimental results were obtained: As shown in figure 9, the function execution time does not float much with the change of memory allocation, which leads to the conclusion that the MapReduce framework is not a memory-sensitive computing framework.

For processor parameters, higher vCPU specifications result in significantly shorter execution times.

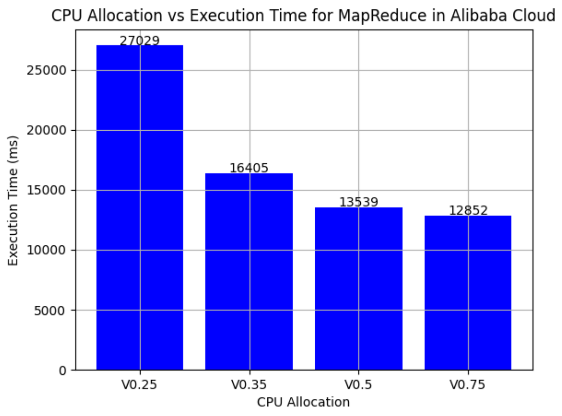

Figure 9. The Relationship with vCPU and execution Time

4. Discussion

4.1. Shortcomings

There are some shortcomings in the serverless based MapReduce framework for functions to deal with relevant experimental problems. The following is a detailed analysis of these shortcomings and possible directions for improvement.

Problems with word filtering: The word filtering part of the function only determines words by determining whether they are all letters, which results in all hyphenated words or other split words with punctuation being filtered out and resulting in false counts. In addition, the function's running time is slightly increased by repeating the word traversal twice to convert lowercase letters and perform other filtering operations.

Repeated communication, communication time is too long: in the reduce phase, every time when importing the processing results of two groups of maps, we need to download the file once, resulting in a total of 26 downloads and 26 uploads after traversing all the maps, which leads to an increase in communication delays and an increase in the likelihood of failures.

Inadequate design, need to be combined with the actual: in the experiment, the paper shorten the running time of the task by increasing the number of groups in each of the Map and Reduce phases, but the increase in the number of threads caused by the increase in the number of groups also leads to an increase in the load on the CPU, which results in the crash of the programme running.

4.2. Directions for Subsequent Improvement

Solve the word filter method:

① Combine operations such as converting words to lowercase, removing punctuation, splitting words (if needed), and filtering special words into a single loop to increase what is processed for words in each loop and reduce the total number of loops.

② User can customise the input line separator to split words more accurately. This avoids incorrectly treating hyphens and neighbouring words as one word. Add filtering design for special words such as hyphens.

Solve the problem of excessive communication time:

① Streaming: consider using streaming processing instead of file-based processing. In this way, the results of the Map task can be sent to the Reduce task in real time without having to write to a file first.

② Reduce the amount of data transferred: optimize the data processing logic to reduce the amount of data transferred between Map and Reduce tasks. For example, more data cleansing and transformation can be performed in the Map phase to reduce the amount of data that needs to be transferred.

③ Optimize file transfers: If you have to use file transfers, consider the following optimization measures: Category 1, use a faster network connection. In the second category, compress files to reduce transfer time. In the third category, use multi-threading or asynchronous I/O to process multiple file transfers in parallel.

Solve the problem of high CPU load: according to the size of the data volume and the host performance of the comprehensive consideration of the map and reduce the number of respective group settings, the optimal solution after many tests.

5. Conclusion

Through this research, the paper implements the use of the MapReduce framework for counting word frequencies, and explores the multiple factors that affect the efficiency of the framework's operation as well as optimization methods.

Firstly, the running time of the task is shortened by increasing the number of groups in the map or reduce phases. The number of groups in each of the map and reduce phases is considered according to the size of the data volume and the performance of the host, and the optimal solution is derived from multiple tests.

Secondly, by modifying the framework, this study obtains the ratio of function execution time and communication time, and clarifies the cost of communication, so as to investigate the measures to reduce the proportion of communication time and improve the efficiency. It can be concluded that: as the number of Map and Reduce functions grows, both communication time and total function execution time decrease. The communication time's share of the total duration also shrinks, indicating enhanced operation efficiency.

Thirdly, by using the features of AliCloud, this study monitored the memory consumption of the function. The magnitude of memory consumption was positively correlated with the size of the dataset, i.e., as the size of the dataset increased, the memory consumption increased accordingly.

Finally, this paper manages to investigate the effect of memory consumption and processor parameters on execution time. The changes in function execution time is not significant as the allocated memory capacity increases, so memory allocation is not the main determinant of execution time. However, higher processor parameters lead to shorter execution times under the same conditions of processing.

In the future, optimization and improvement of MapReduce and similar big data processing frameworks will become more urgent and important as the volume of data continues to grow and the computational tasks become more complex. Any small improvement embodied in the massive amount of data is huge. By continuing to delve into such frameworks, enhancing adaptive tuning, while combining with deep learning, improving cross-framework interoperability, etc., new application areas may open up. The MapReduce framework, which is continuously optimized and iterated, is expected to be expected to become a more powerful, flexible, and secure big data processing solution in the future, providing continuous support and development of data-driven applications in various industries.

Authors Contribution

All the authors contributed equally and their names were listed in alphabetical order.

References

[1] Vouloumzidis, K., et al. 2019. A Systematic Literature Review of Serverless Computing: The First 10 Years. Journal of Cloud Computing,7(3), 255–281.

[2] Dean, J., & Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113.

[3] Wang, L., et al. 2018. "A Survey on Serverless Computing: Architecture, Applications, and Security." IEEE Access, vol. 6, pp. 31535–31553.

[4] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. 2010. Spark: Cluster computing with working sets. In 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10).

[5] Verma, A., Cho, B., Zea, N., Gupta, I., & Campbell, R. H. 2013. Breaking the MapReduce stage barrier. Cluster computing, 16, 191-206.

[6] Gorgolewski, K., Burns, C. D., Madison, C., Clark, D., Halchenko, Y. O., Waskom, M. L., & Ghosh, S. S. 2011. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Frontiers in neuroinformatics, 5, 13.

[7] Wang, G., & Chan, C. Y. 2013. Multi-query optimization in mapreduce framework. Proceedings of the VLDB Endowment, 7(3), 145-156.

[8] Tari, Z. 2014. Security and privacy in cloud computing. IEEE Cloud Computing, 1(01), 54-57.

[9] QIAN Wenjun, SHEN Qingni, WU Pengfei, et al. 2022. Progress of privacy protection technology in big data computing environment. Journal of Computing, 45(4), 669-701.

[10] Ibtisum, S., Bazgir, E., Rahman, S. A., & Hossain, S. S. 2023. A comparative analysis of big data processing paradigms: Mapreduce vs. apache spark. World Journal of Advanced Research and Reviews, 20(1), 1089-1098.