MapReduce Word Frequency Statistics Based on Serverless Platform

Yuewei Zhang *

School of information science and engineering, Shenyang University of Technology, Shenyang, Liaoning, China

* Corresponding Author Email: zyw1756934174@outlook.com

Abstract. In the past few years, people's demand of speed and efficiency in extracting information has increased dramatically, which gives rise to the surge in the demand of text data processing. Traditional distributed computing framework based on VMs are limited by some inherent defects such as wasting resource reservation, large overhead, rigid scaling, which further cause bad utilization and poor performance when running on burst workloads. This paper design and implement a Serverless MR system on Alibaba Cloud Function Compute (FC) and Object Storage Service (OSS). The Map and Reduce part is split into independent and event-driven functions, and then automatically executed and allocated across FC instances. The experiment results on text word frequency counting show that the system is efficient in elasticity and cost. Especially, it's efficient in lightweight applications with high concurrency and transient workloads. This work provides an efficient and manageable big data processing platform for small and medium teams. Meanwhile, it also provides an example of how to apply Serverless computing in distributed processing model.

Keywords: Serverless; MapReduce; Alibaba Cloud Function Compute; Word Frequency Statistics.

1. Introduction

The rise of cloud computing and artificial intelligence has necessitated advanced data processing frameworks capable of handling vast data volumes. In response, the MapReduce programming model was developed to effectively manage and process "Big Data" [1]. As a classic parallel programming model, it has greatly simplified the computing process for large-scale data sets since it was proposed by Google in 2004. Its core idea of "divide and conquer" distributes the computing tasks of large-scale data sets to a cluster composed of numerous ordinary computing devices through two stages - Map and Reduce, ultimately summarizing and outputting the results, significantly reducing the complexity of developing distributed programs. However, the traditional implementation of MapReduce relies on a virtual machine environment that needs to be continuously maintained. When dealing with non-persistent and fluctuating computing tasks, it has problems such as low resource utilization, high costs, and complex operation and maintenance.

The existing Serverless platforms that use containers and micro virtual machines as the runtime environment for functions can cause significant startup delays, thereby hindering the performance and scalability of the executed functions [2]. So, more and more researchers tend to use Serverless computing. Serverless computing is a cloud computing paradigm that frees developers to focus on creating business logic rather than managing the underlying infrastructure. This enables developers to innovate and provide customers with application functions more quickly [3]. As a new type of cloud-native computing model, it completely delegates the management, expansion, and operation and maintenance of infrastructure to the cloud platform [4]. Developers only need to focus on the function code and pay according to the actual execution time and resource consumption [4]. In short, Serverless architecture is a way to build software based on the cloud without dealing with the daily responsibilities such as patch protection, capability control, load balancing, and expansion [4].

If these two can be combined, based on the Alibaba Cloud Serverless platform, using the MapReduce model, and designing and implementing a classic word frequency statistics program, the high resource utilization and cost-effective model of the Serverless platform can be fully exploited. The advantages of Serverless computing can be maximized, big data jobs can be completed more

quickly, and costs can be reduced by supporting MapReduce's big data computing capabilities [5]. Therefore, this research aims to explore innovative integration solutions between the classic MapReduce model and modern cloud-native architectures. Through the Alibaba Cloud Function Compute platform, the Mapper and Reducer of the MapReduce model are decomposed into two independent functions and using the classic “word frequency statistics” task as a case, the system implementation and performance analysis are carried out.

2. Method of MapReduce Words Frequency Statistics

2.1. The Entire Process of the System

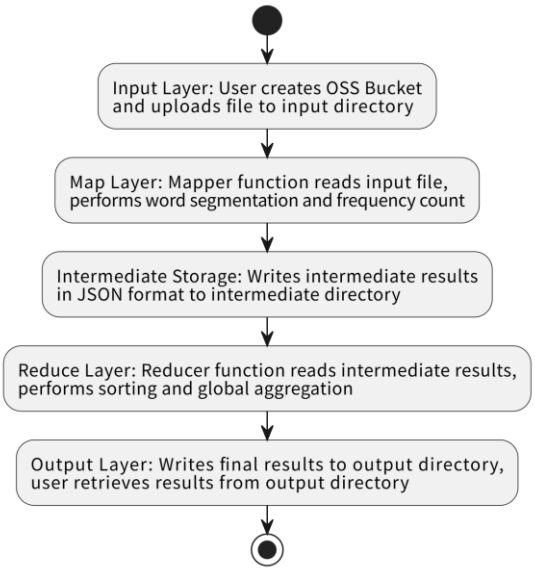


Fig 1. Hole process of MapReduce word frequency statistics based on the Serverless platform (Picture credit: Original).

MapReduce efficiently executes programs on large clusters by utilizing the concept of parallelism [6]. The MapReduce programming approach can divide large-scale data processing tasks into smaller, independent tasks, run those tasks concurrently on server clusters, and then integrate the outputs of those tasks to get the final result [7]. These researchers first created two independent functions, namely the Mapper function and the Reducer function, on the Alibaba Cloud Serverless cloud service platform. Then, we created three directories in the Bucket of the Object Storage Service (OSS), which are called input, intermediate, and output. Users uploaded the text files to be processed to the input directory in the OSS Bucket. Subsequently, the Map stage was automatically triggered by the Mapper function. This function reads the text from input directory, line by line do word segmentation and cleaning, and outputs the statistics results as key-value pair of words. The system stored these statistical results in the OSS intermediate directory, so there is an intermediate storage layer. To support the interoperable and efficient data exchange between decoupled Map and Reduce stages, the system serializes the intermediate results in JSON format, and the structured format enables the intermediate data can be parsed and handled efficiently by the Reducer functions in Reduce stage. In the Reduce stage, reducer receives the intermediate data, sort and final aggregation over the entire data set. Finally, the statistics results will be write into output directory, users could directly collect the results from output directory. With this Serverless Mapreduce computation system users can efficiently get the word frequency statistics. The whole process is shown in Fig. 1.

2.2. Implementation of the Map Layer

In the Map phase, the following mappers are configured by the researcher on Alibaba Cloud Function Compute. Triggered by this event, multiple mappers are started automatically when the user

uploads more text files into the OSS input directory. Read the fragment of the file assigned to this mapper. Parse the text line by line, this includes lower/upper case and remove punctuation. Perform simple word segment and count this include simple word segmentation and count. Output the intermediate key value pair as JSON format and save it in the OSS intermediate layer. This event-driven way of processing data pipelines naturally fits well into the pay-per-use Serverless computing model. That is, more computational resources are automatically allocated as more upload events are triggered by users, and no idle computing resources are charged, which makes the whole solution more cost-effective.

2.3. Implementation of the Reduce Layer

In Reduce stage, the Reducer function reads all the intermediate result files from the intermediate directory of OSS. To do so, it needs to be sorted and aggregated globally by the result from all the Mappers. Finally, the Reduce function sums up the count of the same word and outputs the total number of times each word appearing in the whole novel. Finally, the results are saved in a structured format such as CSV or JSON to the OSS output directory, and it is directly accessible to users. Reduce process elasticity and cost control improvements: Run Reduce process as Serverless function. Perfect use case for computing tasks with clear peak features.

2.4. Data Flow and Storage Design

The whole system is based on OSS. The data flow and storage order are very clear. The data of input parameter “input” are stored in the “input” folder. The intermediate file is stored in the “intermediate” directory. The final output parameter “output” is stored in the “output” directory. This design reduces coupling between functions and makes the function more scalable and maintainable. The whole process is an event. The event of file upload triggers the Mapper. When the intermediate file is generated, the event of reducing is automatically triggered. This is a whole process of automated reduction. There is I/O delay between the application and external storage, but this can be solved by two methods: regional deployment optimization and compression of intermediate data. This solves the network transmission bottleneck.

2.5. Cold Start Problem and Pre-set Concurrency

Serverless computing is a new cloud computing model that relieves the burden of infrastructure management for developers and provides cost saving and scalability [8]. However, cold start latency and the difficulty in realizing complex business logic with workflows are the of serverless computing [8]. However, the cold start latency of MapReduce tasks affects the running performance of short MapReduce tasks on Serverless architectures. The problem happens in the function initialization phase. The cold start latency of this phase occupies hundreds of milliseconds to seconds. For a short MapReduce task, such as processing a small text slice, the overhead may account for a large proportion of or dominate the total running time of the task.

Therefore, the cold start problem is alleviated by using the pre-set concurrency (Provisioned Concurrency) strategy in this study. For example, the instance number of the pre-set concurrency Mapper function in word frequency statistics is configured as 500. The cloud service provider will prepare the corresponding number of function execution environments in advance. When a large number of file upload events trigger function calls simultaneously, the first 500 requests can be directly routed to the ready environment, thus completely avoiding cold starts and achieving the same response performance as a hot start. But if the number of concurrent requests exceeds the pre-set quantity, the excess part will still be processed according to the conventional on-demand expansion mechanism and may again encounter cold starts.

This method effectively eliminates cold start phenomena within the specified concurrency range, significantly improving the performance of short tasks. However, pre-set instances will continuously incur costs, even if no requests are processed, leading to an increase in costs; setting the quantity too high will cause resource waste, and setting it too low will not be able to completely avoid cold starts.

2.6. Network and I/O Optimization

During the subsequent experiments, considering that the system might encounter network I/O performance bottlenecks due to its reliance on OSS for intermediate data exchange, this paper specifically adopted the deployment of Mapper and Reducer functions in the availability zone of the same region as OSS, in order to utilize the internal network channel of the cloud platform to reduce the basic latency; meanwhile, the researchers also adopted a compression algorithm to compress the intermediate data, reducing the amount of transmitted data and further improving the processing efficiency. In the future, it is possible to explore the use of memory-based storage or caching services such as Redis as an intermediate data layer, which will further reduce I/O waiting time and enhance the overall system throughput.

3. Performance Optimization and Analysis

3.1. Parallel Optimization of Mapper and Reducer

Parallel computing means many compute-intensive tasks or processes are executed simultaneously [9]. This study first created a Mapper function to split a large file into four smaller files for processing. The running time at this point was 1.585 seconds. Then, researchers added another Mapper function, allowing one Mapper to handle two smaller files. The running times of the two Mappers were 664 milliseconds and 730 milliseconds respectively, and the time of the Reducer was reduced from 632 milliseconds to 280 milliseconds. These data indicate that by splitting a large file into multiple smaller files and starting multiple Mapper instances to process them in parallel, the time consumed in the Map phase and the Reduce phase can be significantly shortened. However, the number of Mappers is not necessarily the more the better. When researchers increased the number of Mappers to four and let one Mapper handle a small file, the time of the Mapper was indeed shortened, but the time of the Reduce phase increased from 280 milliseconds to 463 milliseconds. This shows that increasing the number of Mappers can indeed improve parallelism, but too many Mappers will lead to an increase in the number of intermediate files, thereby prolonging the time for the Reduce processing. In order to reduce the delay in Reduce stage due to running of four Mapers, researchers run two Reducer in parallel and also implemented parallel execution during Reduce stage to divide the processing of large number of intermediate files which results in decrease in aggregation time. The specific data is shown in Table 1.

Table 1. Performance period of different quantities of Mapper and Reducer

The quantity of Mapper and Reducer	Performance period of Mapper	Performance period of Reducer
One Mapper one Reducer	1.585 milliseconds	632 milliseconds
Two Mapper one Reducer	664 milliseconds 730 milliseconds	280 milliseconds
Four Mapper one Reducer	269 milliseconds 338 milliseconds 276 milliseconds 485 milliseconds	463 milliseconds
Two mapper two Reducer	664 milliseconds 730 milliseconds	259 milliseconds

3.2. Resource Allocation Optimization

In the Serverless architecture, the configuration of memory and CPU directly affects the execution efficiency of functions. Under the same configuration, the number of CPU cores has a significant impact on function response time: more cores result in shorter response times, while increasing memory typically comes with enhanced CPU performance, which further reduces function execution time [10]. More memory will also lead to better CPU performance and further decrease function execution time. By repeated experiments, this paper finds that increasing memory from 512MB to

1GB and CPU from 0.35vCPU to 1vCPU can decrease execution time from 463ms to 396ms. As shown in Table 2, finding the optimal resource configuration to support performance with less cost is crucial. People can automatically apply the parameter-tuning mechanism to recommend resource configuration for different tasks. For example, for I/O-intensive tasks, people can decrease CPU allocation; for computation-intensive tasks, people can increase memory to CPU ratio.

Table 2. Performance period of different resource allocation.

Memory usage	CPU Configuration	Memory configuration	Performance period
500MB	0.35vCPU	512MB	463 milliseconds
500MB	1vCPU	1GB	396 milliseconds

4. Conclusion

This work shows how to apply classic MapReduce into Serverless architecture in the cloud. By splitting the classical MapReduce into independent functions and triggering them by events, and using OSS as the intermediate data storage, the whole system inherently follows the Serverless characteristic of elastic scaling, fine-grained billing, and easy maintenance. The experimental evaluation shows that it is a good choice for simple event-driven and intermittent processing tasks. Compared with traditional clusters, it can avoid the problem of imperfect resource utilization and slow response speed in a flexible way.

This work has some limitations. Firstly, compared with the classical MapReduce, the system has a dependency on external OSS as the intermediate data storage, which may cause certain I/O latency. It may be a bottleneck for more complex computing modes. Secondly, the response speed of ultra-short tasks is still limited by the cold start problem. It can be alleviated by pre-set concurrency. Thirdly, the system is based on word frequency statistics scenario, its application effect of other computing modes, especially iterative computing mode and multi-stage computing tasks, need to be further verified.

Although this work still has certain theoretical significance and practical value. Not only did it prove the possibility of using Serverless architecture in distributed data processing, but it also proposed a high-efficiency big data processing system for small teams.

In the future, researchers will try to introduce a more efficient data intermediate layer to reduce costs and construct an automated configuration recommendation system. The system can adjust the number of concurrent functions and the size of quotas automatically according to the type of tasks.

If it is possible, this paper hopes they can devote themselves into expanding to more complex application scenarios like distributed machine learning, promoting the automated integration of Serverless and artificial intelligence technology, and further improving the system's performance in terms of intelligence, efficiency and green computing.

Reference

[1] Goyal A. Study on emerging implementations of MapReduce. International Conference on Computing, Communication & Automation. IEEE, 2015, 16-21.

[2] Jiang S, Zeng R, Zhou Y, Lyu M R. Distinguishability-guided Test Program Generation for WebAssembly Runtime Performance Testing. 2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2025, 768-779.

[3] Devineni S, Gorantla B, Shaik I, Kumar V A. Optimizing Cloud Resources using Algorithmic Approaches in Serverless Computing. 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2024, 1-6.

[4] Veuvolu R, Suryadevar A, Vignesh T, Avthu N R. Cloud computing based (Serverless computing) using Serverless architecture for dynamic web hosting and cost optimization. 2023 International Conference on Computer Communication and Informatics (ICCCI). IEEE, 2023, 1-6.

[5] Huang X, Gu R, Huang Y. Towards Efficient Serverless MapReduce Computing on Cloud-Native Platforms. Big Data Mining and Analytics, 2025, 8(3): 575-591.

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

[7] Cai J, Huang K, Liao Z. Efficiency Assessment of MapReduce Algorithm on a Serverless Platform. 2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI). IEEE, 2023, 1201-1207.

[8] Jo M, Kim D, Baek I, Park S. Challenges of Serverless Computing Paradaigm. 2025 International Conference on Information Networking (ICOIN). IEEE, 2025, 707-710.

[9] Barcelona-Pons D, García-López P. Benchmarking parallelism in FaaS platforms. Future Generation Computer Systems, 2021, 124: 268-284.

[10] Chen X, Li X, Zhou Q. Imbalanced Word Counting Using MapReduce in Serverless Platform. 2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI). IEEE, 2023, 1195-1200.