# Word Count Frequency Statistics Application of MapReduce Based on Alibaba Cloud FC

Jingqi Zhang

{ zjq2555@smail.seig.edu.cn }

Institute of Electronic Information and Control Engineering,Guangzhou University of Software, Guangdong, China

**Abstract.** The core part of distributed text processing is the MapReduce framework. Alibaba Cloud Function Compute and Object Storage Service can realize its serverless deployment, but the parameter matching rules for data sharding and memory configuration still stay unclear. This study builds a MapReduce distributed architecture based on Alibaba Cloud FC+OSS, and uses English text as test data. Therefore, three groups of controlled variable experiments are designed for exploring the influence of shard size and memory configuration on word frequency counting efficiency. Thus, experiments prove the architecture's feasibility with 100 percent accuracy, and find out the optimal parameter combination is 50MB shards + 1GB memory + 4 concurrency levels—this combination can get 20 percent efficiency improvement without extra costs. Hence, this study makes clear the shared-memory matching relationship, and provides experimental references for parameter configuration in the distributed data processing of Alibaba Cloud FC.

**Keywords:** MapReduce; Alibaba Cloud FC; OSS; Term Frequency Counting; Parameter Optimization

## 1 Introduction

The non-stop growth of text information has pushed forward the progress of distributed data processing technologies. The MapReduce framework, which is based on its divide-and-conquer core rule, has become the mainstream solution for distributed text processing; its high fault tolerance and expandability have been widely proven in large-scale data processing situations [1, 2]. Therefore, MapReduce makes large-scale data processing easier by splitting tasks into parallel-executable subtasks, and this technology has been widely used in various big data scenarios [3]. Traditional MapReduce arrangement relies on physical servers or virtual machines, which need professional staff to conduct management and resource adjustment, thus causing a low dynamic resource usage rate, high running expenses, and obvious resource waste [4].

Serverless architecture allows for on-demand distribution of computing resources, rising up as a new pattern to deal with the limitations of traditional deployment [5, 6]. Alibaba Cloud FC,

ICIAAI 2026, May 22-24, Singapore, Singapore

DOI 10.4108/eai.22-5-2026.2365326

together with high reliability and large-capacity storage features of OSS, offers a new lightweight deployment route for the MapReduce framework. Therefore, this combination unlinks computing and storage, thus greatly increasing the cost optimization space of resource utilization [7]. However, the current study on combining serverless architecture with the MapReduce framework is still in the exploration stage, hence lacking clear matching rules for how key parameters like data shard size, memory configuration, and concurrency influence processing efficiency and costs [7, 8].

In distributed word frequency counting programs, unsuitable parameter matchings will easily bring about memory overflow, low work efficiency, and high expense. The scientific property of data sharding strategies directly decides the whole efficiency of distributed computing, as improper sharding can result in unbalanced task distribution and lowered parallel efficiency [9]. At the same time, the cold start delay of serverless architecture is still a key factor that limits its application in distributed data processing. The extra delay caused by function instance initialization has an influence on the real-time performance of the processing procedure, and effective relieving methods are still in the process of exploration [10, 11].

This research concentrates on distributed word frequency statistics of the English language text. It constructs a MapReduce distributed framework based on Alibaba Cloud FC+OSS, finishes function establishment, arrangement, and authority setting, and designs three groups of controlled variable experiments. Therefore, fixing the concurrency number at 4 and carrying out three repeated tests to decrease mistake occurrences, it investigates how data shard size and memory configuration influence system operation efficiency and stability, hence finding the best parameter combination that balances efficiency, stability, and cost, and verifies that this serverless framework is feasible.

## 2 Research Methodology

### 2.1 Overall process

Centered on the divide-and-conquer principle of MapReduce, this research takes Alibaba Cloud OSS as the inter-function data transmission and storage medium, so as to construct a distributed word frequency counting process which is based on FC+OSS. OSS triggers can automatically set off Map and Reduce functions; the concrete process is displayed in Table 1. This process makes reference to the classic distributed computing data flow model under serverless architecture, thus decoupling computing nodes via intermediate storage media, which is in accordance with the parallel processing logic that MapReduce advocates.

**Table 1.** Overall Process of MapReduce Word Frequency Statistics Based on Alibaba Cloud FC+OSS.

| Process Stage | Core Operation | Execution Subject | Data Storage Medium |
|---|---|---|---|
| 1. Dataset Preparation | Select a unified English text to construct the experimental basic dataset | Manual | Local Storage |
| 2. Text Sharding | Split the basic dataset into text shards of different sizes | Manual | Local Storage |

| | | | |
|---|---|---|---|
| 3. Map Local Statistics | Upload shards so as to set off the Map function and finish local word frequency statistics targeted at one single shard | Alibaba Cloud FC (Map Function) | Alibaba Cloud OSS (Shard Files) |
| 4. OSS Intermediate Storage | Generate intermediate files of local word frequency statistics and store them | Alibaba Cloud FC (Map Function) | Alibaba Cloud OSS (Intermediate Data Directory) |
| 5. Reduce Global Aggregation | Intermediate files trigger the Reduce function to merge and aggregate global word frequencies | Alibaba Cloud FC (Reduce Function) | Alibaba Cloud OSS (Intermediate Files) |
| 6. Word Frequency Result Output | Generate global word frequency statistics results and perform persistent storage | Alibaba Cloud FC (Reduce Function) | Alibaba Cloud OSS (Result Directory) |

### 2.2 Function and trigger design

Both Map and Reduce functions are constructed through Alibaba Cloud FC's blank templates, operating as event functions on Python 3.9 and being automatically triggered by OSS triggers. As one kind of lightweight development language, Python can effectively lower the cold start long-tail delay of serverless architecture, thus adapting itself to the lightweight computing demands of word frequency statistics. The Map function is bound to an OSS trigger and connected with a designated OSS Bucket, carrying out monitoring of file upload events. When shards are uploaded, it will automatically operate to read shards, conduct preprocessing and text tokenization, count single-shard word frequency to form key-value pairs, and save local results into the intermediate data directory of OSS.

The OSS trigger belonging to the Reduce function carries out monitoring of file establishment events inside the OSS intermediate data directory. After local statistics files are stored, it will automatically operate to travel through and read all local word frequency files, combine key-value pairs that belong to the same word, compute total frequency, and at the end produce global word frequency results and save them into the designated result directory of OSS. This process achieves data transmission between the Map and Reduce phases via OSS, thus avoiding node communication complexity of traditional MapReduce clusters, which is in line with MapReduce's distributed data management thoughts.

### 2.3 Experimental design method

In order to design comparative experiments, this study makes use of the controlled variable method. After carrying out preliminary adaptability testing, concurrency is fixed at 4 to satisfy the needs of text sharding processing. This paper designs three groups of comparative experiments, and each of them is repeated three times for the purpose of averaging results and eliminating random errors; four core data points are recorded, including total running time, cold start delay, running status, and word frequency counting accuracy.

Group 1 fixes the memory amount at 1GB, makes use of 30MB×2, 50MB×2, 80MB×2 shard sizes to carry out exploration on the shard size on system efficiency and stability. Group 2

fixes the shard size at 50MB×2, configures memory as 512MB, 1GB, 2GB to verify the speed promotion effect brought by memory. Group 3 takes 80MB large shards as research target, matches them with 512MB, 1GB, 2GB memory to explore the adaptive relationship between large shards and memory, and solve memory overflow problems caused by large-scale shards. The experimental design references parameter optimization thoughts in MapReduce performance research.

### 2.4 Permissions and environment configuration method

Before carrying out experiments, finish the basic configuration of Alibaba Cloud FC and OSS, activate these two services in sequence, acquire AccessKey ID and Secret, and finish permission verification for FC functions to access OSS. Therefore, to avoid permission access mistakes, add AliyunOSSFullAccess system permissions to Map and Reduce functions, thus ensuring they possess full reading and writing permissions for OSS files, realizing normal data interaction between FC and OSS. Hence, this permission configuration scheme is the standard configuration method for collaborative application of Alibaba Cloud FC and OSS, ensuring stability of the distributed computing environment.

## 3 Experiment

### 3.1 Experimental environment

The experimental environment is composed of four components: cloud services, development environment, test data, and auxiliary tools. All configurations, whose key details are displayed in Table 2, must guarantee the experiment's consistency, comparability, and reproducibility. Alibaba Cloud FC has officially recommended Python 3.9 as its adaptive running environment, and this environment can support core word frequency statistic operations, such as text tokenization and file operations, in a good manner. In accordance with the standard configuration of big data processing based on MapReduce, and have built the experimental environment.

**Table 2.** Key Configuration Information of the Experimental Environment

| Configuration Type | Specific Configuration Content | Core Function |
|---|---|---|
| Cloud Services | Alibaba Cloud Function Compute (FC), Alibaba Cloud Object Storage Service (OSS) | FC carries out distributed function computation work, and OSS performs the storage as well as transmission of data and results |
| Development Environment | Python 3.9 | Function development, carrying out adaptation to the FC runtime environment, and providing support for text processing operations |
| Test Data | A single English text split into: 30MB×2, 50MB×2, 80MB×2 | Ensure data consistency and explore the impact of shard size on processing |
| Auxiliary Tools | Alibaba Cloud FC/OSS Console | Service management, function configuration, OSS trigger |

| | |
|---|---|
| | binding, data upload, and result recording |

### 3.2 Evaluation metrics

This work chooses four core measurement indices to appraise architecture performance from efficiency, stability, and accuracy aspects. Table 3 displays each index's definition and mathematical computing formula, so as to simplify the calculation process. Among them, cold start delay is one core evaluation index for distributed processing in serverless architecture; the official measurement standard of Alibaba Cloud FC cold start delay is referenced for its statistical method, and the calculation logic of word frequency counting accuracy follows the general evaluation method of distributed word frequency counting.

**Table 3.** Experimental Evaluation Metrics and Calculation Formulas.

| Evaluation Metric | Unit | Metric Definition | Mathematical Calculation Formula |
|---|---|---|---|
| Total Runtime | Second (s) | The total time from uploading text shards to OSS to generating global word frequency results | Ttotal=Tresult−Tupload<br>Tupload: Timestamp of shard up load<br>Tresult: Timestamp of global result generation |
| Cold Start Delay | Second (s) | The extra time delay that comes from FC instance initialization, starting from function triggering moment up to actual code execution beginning | Tdelay=Trun−Ttrigger<br>Ttrigger: Timestamp of function triggering<br>Trun: Timestamp of actual code execution |
| Operational Status | - | Evaluate the operational stability of the system, divided into Stable/Failed categories | No formula. Stable status = Finish all data processing works without any mistakes; Failed status = Stop operation process because of permission problems, memory overflow and other related situations |
| Word Frequency Statistics Accuracy | Percentage (%) | The overlapping degree between global word frequency outcomes produced by the architecture and manual statistics outcomes | A=Nmatch/Ntotal ×100%<br>Nmatch: Number of words with consistent frequency statistics<br>Ntotal: Total number of words participating in statistics |

### 3.3 Experimental results and analysis

**Basic Experimental Results.** Baseline experiments finish the double verification of experimental environment effectiveness and architecture function feasibility. In the course of the experiment, there is no permission access error between FC and OSS, and the triggering, running, and data interaction processes of Map and Reduce functions are stable with no abnormal interruption. Functional verification displays that the Map function can exactly read text shards and finish local word frequency counting, with intermediate results normally stored in the pointed OSS directory; the Reduce function can traverse and read all intermediate files, exactly finish global word frequency aggregation, and the statistical results are fully consistent with manual counting, with accuracy reaching 100%. Therefore, this verifies the technical feasibility of the architecture in distributed word frequency counting, hence proving that

serverless architecture can effectively carry out the core divide-and-conquer logic of MapReduce.

**Comparison of Optimized Experimental Results.** All three groups of optimized experiments keep 100% correctness of word frequency counting, with differences showing in total operation time, cold startup delay, and operation state. After the averaging process, the experiment-related data are concluded into Table 4.

**Table 4.** Comparison of Optimization Experiment Results.

| Partition Size | Memory Configuration | Total Runtime (s) | Cold Start Delay (s) | Operational Status | Word Frequency Count Accuracy (%) |
|---|---|---|---|---|---|
| 30MB×2 | 1GB | 45 | 7 | Stable | 100 |
| 50MB×2 | 1GB | 60 | 7 | Stable | 100 |
| 80MB×2 | 1GB | 85 | 8 | Stable | 100 |
| 50MB×2 | 512MB | 75 | 10 | Stable | 100 |
| 50MB×2 | 1GB | 60 | 7 | Stable | 100 |
| 50MB×2 | 2GB | 58 | 6 | Stable (Low Cost-Performance Ratio) | 100 |
| 80MB×2 | 512MB | - | - | Memory Overflow Error | - |
| 80MB×2 | 1GB | 85 | 8 | Stable | 100 |
| 80MB×2 | 2GB | 80 | 7 | Stable (High Cost) | 100 |

Tests under fixed memory and different shard sizes display that total system operation time goes up in a notable way as the shard size becomes larger, whereas cold start delay remains stable at 7 to 8 seconds. Therefore, this is because bigger shards mean the Map function must process more text content at the same time, which prolongs single-function operation time, shard size has no clear effect on FC instance initialization process. This outcome is in accordance with MapReduce's performance features during large-scale data processing. Tests applying fixed data shards and changeable memory setups demonstrate that memory enlargement can usefully cut down overall running duration and cold start latency, yet there exists a diminishing return phenomenon: when memory grows from 512MB to 1GB, overall running time drops from 75 seconds to 60 seconds, latency reduces from 10 seconds to 7 seconds, bringing 20% processing efficiency promotion; when memory rises from 1GB to 2GB, time and latency have tiny decrease, whereas cloud computing resource cost increases in a marked manner, which lowers system cost-performance. This is in accordance with the resource allocation optimization principles within serverless computing. Experiments which use 80MB large shards to match various memories display that the matching degree between large shards and memory is of crucial importance to system stability: when 512MB memory processes 80MB shards, memory overflow errors will be caused, thus making task completion impossible; if memory is increased to 1GB, the overflow problem can be solved and system

operation stability can be ensured; if memory is increased to 2GB, running time will be shortened in a slight way, but resource costs will be increased in a significant way, and no obvious efficiency improvement value can be seen. Therefore, this result verifies the core principle of matching shards and computing resources in distributed data processing.

**Analysis of Optimal Parameter Combinations.** After all-around consideration of processing efficiency, operation stability, and resource expense, this study fixes the optimal parameter set as 50MB shards + 1GB memory + 4 concurrency degrees. Under this set, total system operation time is 60 seconds, cold start delay is 7 seconds, it runs stably without mistakes, and the word frequency counting accuracy reaches 100%. Compared with the 50MB+512MB set, therefore, it lifts processing efficiency by 20% without extra cost, and cuts cold start delay notably; compared with 80MB big shards, it prevents resource cost rise brought by large memory; compared with 30MB small shards, it decreases Map function running times, thus being more fit for large-scale text distributed processing scenes. For MapReduce parameter tuning in serverless environments, this optimal parameter combination offers a practical reference.

## 4 Conclusions

This study constructs a MapReduce distributed structure that is based on Alibaba Cloud FC+OSS, and finishes comparative experiments that take English text word frequency calculation as a research case. Therefore, it verifies this serverless architecture's feasibility in distributed text processing; thus, it can stably fulfill local word frequency counting and global gathering of text fragments, with 100% word frequency counting correctness, hence providing one feasible resolution for large-scale text's serverless processing.

Experiments make clear the impact rules of parameters to system performance: thus, total running time rises with the growth of shard size, whereas cold start delay receives smaller influence from shard size; increasing memory can effectively lift processing efficiency and cut cold start delay, but when memory configuration surpasses 1GB, the improvement effect shows marginal diminishing, and extra resource costs are brought about; large-size data shards need to be paired with enough function memory, otherwise, memory overflow is probable to cause system operation interruption, which verifies the core principle of shard and computing resource matching in distributed data processing. Hence, the optimal parameter combination of 50MB shards + 1GB memory + 4 concurrency levels determined in this study realizes a triple balance among processing efficiency, running stability, and resource costs.

This study discovers two restrictions existing in the architecture: Alibaba Cloud FC's serverless architecture has built-in cold start delay, which shows more clearly when the function is operated for the first time. Hence, this is an inherent trouble brought by the elastic scaling features of serverless architecture, and at present, people can optimize it via methods like instance preheating; the Shuffle stage of the architecture uses OSS directory automatic grouping, hence this may influence the file reading and word frequency aggregation efficiency of the Reduce function in situations with big data volumes. Therefore, to solve the above problems, future research can optimize cold start delay through FC's "function instance preheating" function; at the same time, redesign the grouping rules of the Shuffle stage to raise the processing efficiency of the Reduce function, learning from the hash grouping

optimization strategy in distributed data sharding; in addition, expand the scale and type of test data to explore the parameter matching rules of the architecture in different text scenarios, further enhancing its practical application value.

This study's results offer practical references for Alibaba Cloud FC's application in the distributed data processing field, verify the technical adaptability existing between serverless architecture and the MapReduce framework, clarify optimal parameter configuration within distributed word frequency counting situations, and provide experimental foundation for serverless deployment of other distributed data processing tasks that are based on the MapReduce framework. Therefore, it exerts a positive function on boosting serverless architecture's application in the distributed data processing field; hence, it also supplies new practical thoughts for cloud-native technology's application in the text processing field.

## References


[1] Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM. Vol. 51, No. 1, pp. 107-113 (2008)

[2] Li, F., Ooi, B. C., Özsu, M. T., Wu, S.: Distributed data management using MapReduce. ACM Computing Surveys (CSUR). Vol. 46, No. 3, pp. 1-42 (2014)

[3] Thorat, D. B.: Enhanced genetic algorithm to reduce makespan of multiple jobs in map-reduce application on serverless platform. MSc research project. National College of Ireland, Dublin, Ireland (2020)

[4] Grolinger, K., Hayes, M., Higashino, W. A., et al.: Challenges for mapreduce in big data. 2014 IEEE World Congress on Services. pp. 182-189 (2014)

[5] Kumari, A., Sahoo, B., Behera, R. K.: Mitigating cold-start delay using warm-start containers in serverless platform. 2022 IEEE 19th India Council International Conference (INDICON). pp. 1-6 (2022)

[6] Li, Y., Lin, Y., Wang, Y., et al.: Serverless computing: state-of-the-art, challenges and opportunities. IEEE Transactions on Services Computing. Vol. 16, No. 2, pp. 1522-1539 (2022)

[7] Adzic, G., Chatley, R.: Serverless computing: economic and architectural impact. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 884-889 (2017)

[8] Karloff, H., Suri, S., Vassilvitskii, S.: A model of computation for mapreduce. Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 938-948 (2010)

[9] Glushkova, D., Jovanovic, P., Abelló, A.: Mapreduce performance model for Hadoop 2.x. Information Systems. Vol. 79, pp. 32-43 (2019)

[10] Huang, X., Gu, R., Huang, Y.: Towards efficient serverless MapReduce computing on cloud-native platforms. Big Data Mining and Analytics. Vol. 8, No. 3, pp. 575-591 (2025)

[11] Abdalla, H. B., Kumar, Y., Zhao, Y., et al.: A comprehensive survey of MapReduce models for processing big data. Big Data and Cognitive Computing. Vol. 9, No. 4, pp. 77 (2025)