Deploying a MapReduce Distributed Computing System on a Serverless Platform

Mohan Zhang

{Zmh3201833422@outlook.com}

Big Data Institute, Guizhou Institute of Technology, Guiyang, Guizhou, China

Abstract. As serverless entering clouds computing, MapReduce, a traditional design of massive data processing, is slowly becoming a part of serverless computing engines. This study formulates and deploys a lightweight, elastic, and pay-as-you-go MapReduce distributed computing platform on the open-source serverless platform, OpenWhisk. The model breaks down the classical MapReduce model into serverless functions and relies on an object storage service (MinIO) to accomplish data interchange among functions, which can be dynamic in resource scheduling and task coordination. The findings of this experiment indicate that the system exhibits the sharing of good performance in parallelism and cost-effectiveness in small and medium-scale data processing processes, particularly when they lead to short-term, high-concurrent processing and intermittent computing processes. But when the data processing involves a large amount of data, storage and network communication overheads are the primary bottlenecks. The work has practical references in designing and optimizing distributed computing in a serverless environment.

Keywords: Serverless computing, MapReduce, Distributed system, Cold start.

1 Introduction

Serverless computing is the upcoming level of evolution of the cloud-computing programming models, but it frees the developer of the challenges of infrastructure management [1]. Cloud computation technology has been developed after virtualization and container images for serverless computing. By fully replacing the management of underlying infrastructure with cloud platforms, serverless computing enables developers to concentrate on business logic. A review conducted by Besozzi et al. shows that serverless computing finds more and more applications in the high-performance computing, AI, and big data domains, which indicates its potential in data-intensive applications [2], with more efficient resource usage and reduced operational expenses. Rad and Ghobaei-Arani systematically categorised the information pipeline strategies in serverless computing into a set of thumb-based, machine learning-based, and framework-based data strategies, and gave a framework upon which serverless data processing architectures can be understood [3]. One conventional distributed data processing

ICIAAI 2026, May 22-24, Singapore, Singapore

DOI 10.4108/eai.22-5-2026.2365238

model that has found extensive applications in big data processing is MapReduce popularized by Google in 2004. Nevertheless, the conventional MapReduce systems like Hadoop utilize fixed resources of a cluster, which have problems like poor resource utilization, intricate operation and maintenance and poor scalability. Huang et al. found three important issues with executing MapReduce jobs on serverless systems, including poor scheduling policies, the prohibitive cost of reading Shuffle index data on cloud storage, and a fat tail effect of cloud storage I/O latency. Their optimization plans cut the average job execution time by 25.6 percent [4]. The combination of serverless computing and MapReduce has become an area of study over time in recent years. Based on the FaaS model and the high concurrency of the MapReduce, Li et al. found that word frequency statistics task execution would decrease the time-to-run rate at various frequencies, allowing the discovery of the optimal function configuration to run such tasks under selected workloads [5]. To do on-demand execution, elastic scaling and fine-grained control of costs, MapReduce tasks can be divided into a set of serverless functions.

Past works have tried to apply the MapReduce model to the serverless architecture. There is an example with AWS Lambda, which is used to perform distributed computing and still struggles with data exchange and scheduling of tasks. A broad review of cold start delay reduction mechanisms by Ebrahimi et al. demonstrates that despite multiple strategies being suggested, which include function preheating, snapshot-based recovery, and lightweight sandboxes, cold start is a core problem in serverless computing since the functionality of the distributed process of data processing is largely compromised [6]. The aim of this paper is to address systematic design and experimental analysis solutions and optimization directions to the problems mentioned above. Su et al. proposed a workload-aware lifecycle management strategy to reduce cold start overhead and minimise the overhead of workload-intensive serverless applications, which, in the context of data intensive serverless applications, benefits from intelligent resource management to greatly enhance the performance of bursty and sparse workloads [7].

The paper relies on an open-source Apache OpenWhisk platform and works on the designs and implementation of a full Serverless MapReduce framework, with the purpose of finding out how applicable it is in lightweight and short duration and high-concurrency data processing processes.

2 Research Technique

2.1 Overall process

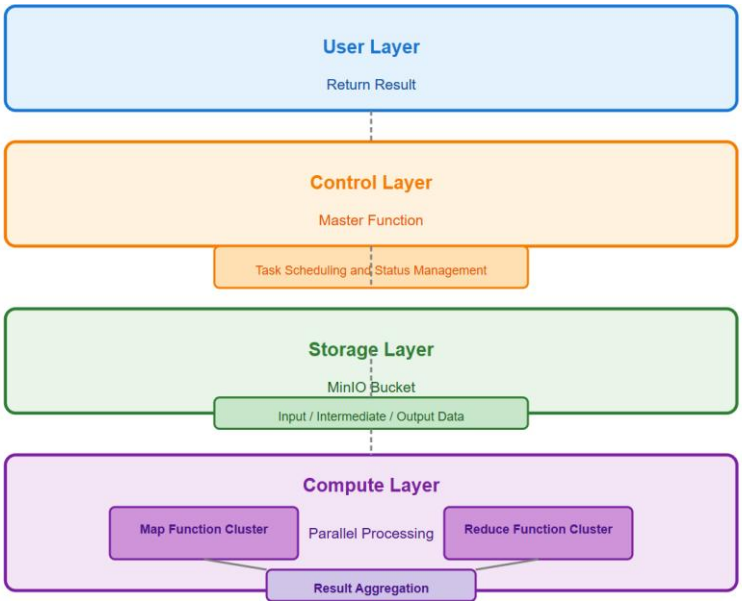

Fig.1. Experimental Procedure (Picture credit: Original)

The general process map is provided in Figure 1. The Serverless MapReduce framework suggested in the current paper is grounded on the OpenWhisk platform and can be divided into the following main components: The Master function, which has task scheduling, data splitting, invoking functions, and aggregating results. The Map functionality cluster is able to run the input data in parallel to create intermediate pairs of key-values. The results of the intermediate can be grouped together and aggregated at the Reduce functionality cluster to produce the final result. Antuzi (2024) introduced an unserverless MapReduce design framework, where the object storage (S3) is utilized and then the data exchange between the Lambda functions is realized in the framework of object storage [8] that can be used as the foundation of the unserverless data processing pipeline. Input data, intermediate results, and final outputs can be stored in MinIO object storage, which is capable of decoupling the data between functions.

2.2 Task execution process

The system receives data processing tasks (including word frequency statistics tasks) from users. These parameters are the whereabouts of input data (MinIO bucket path) as well as the configuration parameters (e.g., the number of Map/Reduce). The Master operation is activated which is that of: reading the details of the shards of the input data; partitioning the information into several Map operations; Time scheduling several Map functions and activation of these functions running in parallel. Each Map function: reads the assigned data shard from MinIO;

executes the Map logic (such as tokenization and counting in word frequency statistics); writes the intermediate results (key-value pairs) to the MinIO storage bucket for the Reduce stage to use. Intermediate data is stored in MinIO and is actively pulled by the Master function or Reduce function; the data is grouped and sorted by key to prepare for the Reduce stage. The Master function triggers multiple Reduce functions; each Reduce function reads the intermediate data belonging to its key group from MinIO, executes the Reduce logic (such as merging counts), and writes the final results to the MinIO storage bucket. The Master function monitors the completion of all Reduce tasks. Summarize the final results and store the output file in MinIO for users to download or further process. All functions complete execution, and the server platform automatically recovers resources; users can view the execution logs, performance indicators, and result files.

3. Experiment

3.1 Dataset

Table 1. Dataset

Project	Description
Type	English text corpus (public dataset)
Task	Word frequency statistics
Scale range	1GB to 10GB (multiple different scales for testing)
Storage location	MinIO object storage bucket (input data bucket)
Format	
Purpose	Text files (such as .txt or similar formats)
Project	Performance of the testing system under different data volumes, such as execution time, cold start impact, scalability, etc.

As shown in Table 1, the type of the dataset is an English text corpus, the task is word frequency statistics, the scale ranges from 1GB to 10GB, and it is applied to multiple different scales for testing. It is stored in the MinIO object storage bucket and describes the text files, which are used to test the performance of the system under different data volumes. References. All references must be in the same format as the ones at the end of this document and the reference list must include all cited literature.

3.2 Experimental environment and experimental procedures

The experimental platform is the Apache OpenWhisk 1.0.0 deployed on a Kubernetes cluster. The programming language used is Python 3.7. The storage service relies on MinIO 2022-06-07. The network setup uses a gigabit internal network environment. Meanwhile, the dataset uses an English text corpus (1GB - 10GB).

This research conducts a Serverless MapReduce experiment based on the OpenWhisk platform. The steps are as follows: First, create three storage buckets - input, intermediate, and output - in the MinIO object storage, and upload an English text dataset of 1-10GB. Then, write and deploy three types of functions: Master, Map, and Reduce. The Master is responsible for task scheduling, while Map and Reduce execute mapping and aggregation logic respectively. Subsequently, trigger the Master function to start the word frequency

statistics task. The system automatically calls the Map function cluster for parallel data processing, and the intermediate results are exchanged through MinIO before being summarized by the Reduce function. During the experiment, monitor the execution status of the functions, cold start delay, and resource usage in real time. Finally, collect and analyze the task duration, scalability, and cost-effectiveness data under different configurations.

3.3 Analysis of experimental results

The experiment employed the word frequency counting (Word Count) task, and the results are shown in Figures 2 and 3 below:

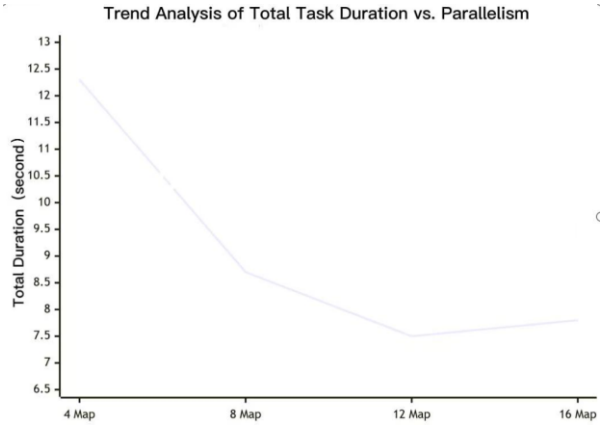

Fig. 2. Trend Analysis of Total Task Duration vs. Parallelism (Picture credit: Original)

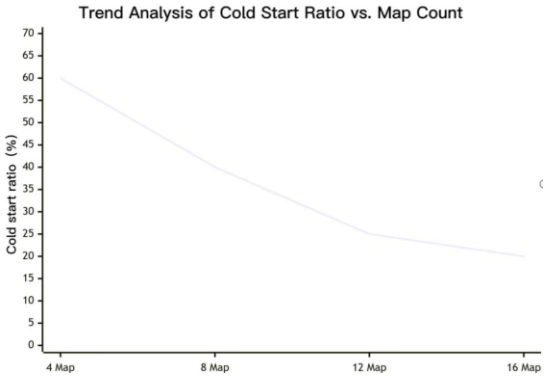

Fig.3. Trend Analysis of Cold Start Ratio vs. Map Count (Picture credit: Original)

Table 2. Experimental Data

Number of Maps	Number of Maps	Total time (seconds)	Cold start ratio
4	2	12.3	60%
8	4	8.7	40%
12	6	7.5	25%
16	8	7.8	20%

Based on the findings in Table 2, the MapReduce system based on Serverless exhibits a high level of elasticity and scalability with cost-effectiveness for small and medium-sized text data of interest. The time used to complete the task tasks reduces as the number of Map functions grows (4 to 12) by 5 seconds to 7.5 seconds with an increment in parallel efficacy and the cold start ratio also decreases (60-25), meaning that the container reuse mechanism has been successful in minimizing the cold start time. Nevertheless, with the addition of the Map number further to 16, the performance time slightly backfires (7.8 seconds), which indicates that the network and storage I/O of the Shuffle stage is a bottleneck, and competition among the resources increases. Also, the memory structure directly affects the speed of function performance; the faster the memory performance, the faster the processing, and the higher the running expenses. The designed Raptor by Exton and Read (2024) is a distributed scheduling service that works alongside OpenWhisk and reduces the cold start execution time down to as low as 80 seconds, steady-state execution time to as low as 10 seconds, and negligible overhead demonstrates that using scheduling can substantially improve serverless platforms serving as a parallel processing model like MapReduce [9].

This paper has managed to deploy a lightweight and auto-scaling Serverless MapReduce system into the OpenWhisk platform and confirm that it is viable and beneficial in short-term, high-concurrent and ad hoc data processing cases. The system has pay-per-use pricing and minimal complexity of operation thus it is appropriate in light big data tasks. The three main challenges of serverless MapReduce reported by Huang et al. (2025) are low scheduling efficiency, costly access to Shuffle index data in cloud storage, and long-tail effect of cloud storage I/O latency. Their optimization solutions eliminated job execution and cost by 25.6% and 17.3% respectively, and gravitated to the Shuffle bottleneck as had been seen in artificial experiments [4]. Further refinement can be done to M-Spin by refining the process of data localization, adding pipeline Shuffle, and preheating techniques to make deep enhancements on the overall performance and efficiency of the large-scale data processing. Li et al. (2024) depicted that in counting word frequency in word frequency counting tasks on serverless platforms, the execution time reduces at varying rates with the number of map and reduce functions and this allows the fatal finding that the optimum combination of functions to be found in special workloads. Their results agree with the experimental results that at a point where parallelism exists, there are diminishing returns [10].

4. Conclusions

The pay-as-you-use model of serverless computing is most appropriate for activities that have high load variances and are executed seldom. The experimental data demonstrate that with the help of the automatic scaling mechanism, the system does not deteriorate, but its performance is stable even with the increase in the amount of data. This is in line with the benefits of FaaS in edge computing and real-time data processing. Similarly, significant cost savings were observed in a log analysis system based on AWS Lambda. This study is based on the OpenWhisk serverless platform and designs and implements a lightweight, elastic-scaling MapReduce distributed computing framework. By constructing a Master scheduling function, Map and Reduce processing function clusters, and combining with MinIO object storage to achieve data exchange, the entire process of task segmentation, parallel processing, and result aggregation was verified. The experiment used an open English text dataset (1-10GB) as input

and performed the word frequency statistics task. The system had good-scale benchmarks of task parallelism with resource elasticity. The findings indicate that one can see that, with a growing number of Map functions, the time spent executing the task is shortened substantially and the ratio of cold start is largely thanks to the reuse of the storage and network I/O, which become bottlenecks, and the execution time returns. The system proves the high cost benefits and easy operation and maintenance behavior in small-scale data processing, which proves the applicability of the Serverless architecture to short-time, high-concurrent computing processes.

References

[1] Jonas, E., et al.: Cloud Programming Simplified: A Berkeley View on Serverless Computing. arXiv preprint arXiv:1902.03383 (2019)

[2] Besozzi, V., Della Bartola, M., Dazzi, P., & Danelutto, M.: High-Performance Serverless Computing: A Systematic Literature Review on Serverless for HPC, AI, and Big Data. IEEE Access, vol. 13, pp. 195611-195656 (2025)

[3] Shojaee Rad, Z., & Ghobaei-Arani, M.: Data pipeline approaches in serverless computing: a taxonomy, review, and research trends. Journal of Big Data, vol. 11, no. 1, pp. 82 (2024)

[4] 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)

[5] Li, H., Lin, B., & Xu, M.: Word Frequency Counting Based on Serverless MapReduce. arXiv preprint arXiv:2601.00380 (2026)

[6] Hao, F., et al.: Customizable function scheduling for serverless computing. SciEngine (2025)

[7] Sang Hongyan et al.: A method for optimizing the cost of MapReduce jobs on a serverless platform. Chinese Invention Patent CN121116559B (2026)

[8] Antuzi, D.: Blazing-Fast Serverless MapReduce Indexer for Apache Solr. Berlin Buzzwords 2024 (2024)

[9] Exton, K., & Read, M.: Raptor: Distributed Scheduling for Serverless Functions. arXiv preprint arXiv:2403.16457 (2024)

[10] Li, H., Lin, B., & Xu, M.: Word Frequency Counting Based on Serverless MapReduce. Proceedings of the 1st International Conference on Engineering Management, Information Technology and Intelligence (EMITI 2024), pp. 40-45 (2024)