Proceedings of the 6th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/68/20241535

Design and implementation of efficient distributed deep learning model inference architecture on serverless computation

Xiaoyang Guo[1,4], Zhe Yang[2], Jie Yi[3]

[1]School of Artificial Intelligence, Nanjing University, Nanjing, China
[2]College of Electronics and Information Engineering, Shenzhen University, Shenzhen, China
[3]International Education College, Zhengzhou University of Light Industry, Zhengzhou, China

[4]guoxy@smail.nju.edu.cn

Abstract. Although the distributed algorithm and architecture have been widely used in the development of traditional large-scale machine model reasoning tasks, but it is unavoidable about the source waste caused by synchronization between learners and participants. However, serverless computing is charged based on usage, which makes it a popular alternative for model inference. This article mainly proposes the architecture and algorithm that can be used for the inference task of the distributed deep learning model and adopts coarse-grained parallelization strategy to solve the problem of high communication cost between Serverless platform functions. The paper tested the performance of the model on the deep learning model and found that the model reasoning based on the Serverless platform can better cope with the requirements and changes, and ensure the stability of the system. It is worth noting that both the data owner and the model owner can benefit from the process of carrying out the machine learning model inference task, but existing solutions cannot satisfied with them privacy requirements, which is a problem needs to think about in the field of machine learning inference tasksbased on the Serverless platform in the future.

Keywords: Serverless Computing, Machine Learning Inference, Parallel Computing.

1. Introduction

In recent years, the development of serverless computing technology is very rapid, and it has gradually developed into a new trend in machine learning. Serverless architecture, as an emerging paradigm of serverless computing, can provide a more flexible and efficient way for machine learning model inference work to build and assign applications. In the face of large-scale machine learning model inference tasks, the research team also needs to pre-configure and manage a large number of computing resources to build GPU data centers when carrying out their work. A GPU data center requires advanced hardware resources at every stage of machine learning to meet its growing development needs and take on workloads [1]. So that is why the traditional machine learning model inference has the problems of low resource utilization, long deployment cycle and high cost. However, MINIONSPL, the first serverless computation-based distributed DRL training framework evaluated on a realistic cluster of popular tasks at OpenAI GYM, showed that this training framework reduced the total training time and high training cost by nearly half compared to traditional machine learning model inference tasks [2]. Therefore, serverless computing is a favorable solution to these problems. Developers just need to focus on the business logic and do not need to care about the underlying server management, which further results in the development of the Serverless platform. Amazon and Alibaba, as major cloud providers, offer inference services by deploying deep learning models, but online inference services are typically delay-critical and resource-intensive, creating tradeoffs between performance and resources. This research work mainly focuses on the insufficient performance of computing functions and high communication delay of Serverless platform [3]. This machine learning model inference research work is deployed on Alibaba Cloud service platform, and uses an architecture similar to Map-Reduce to decompose tasks and integrate results. However, existing solutions cannot satisfy their privacy requirements [4], which is a problem that needs to be considered in the field of machine learning inference tasksbased on the Serverless platform in the future.

2. Related works

In this section, the paper will consider solutions to the large service delay caused by the limited computing resources of the Serveless platform, including CPU and memory, as well as solutions to reduce costs. The paper investigated the design of methods including data segmentation, distributed processing, model partitioning strategies, and data transmission.

The paper first consider ways to reduce the cost of inference services - Chahal et al. [5] Proposed hybrid cloud approach, which utilizes on-premises infrastructure and cloud services to dynamically adjust resource allocation for greater flexibility, scalability, and cost effectiveness.

The paper also learned that the FSD-Inference [6] inference system achieves efficient parallelization of the full connection layer while maintaining high performance under sporadic workloads, and introduces serverless point-to-point data communication as a solution. Furthermore, Gillis [7] has a model service system based on serverless architecture, which performs coarse-grained model partitioning to reduce communication between the main function and sub-functions, thereby avoiding frequent synchronization with other functions.

The paper also learned about the strategies for model partitioning in the serverless framework design for fast and cost-effective deep learning model training implemented by FuncPipe [8]. The goal of model parallelization is to minimize the processing time of one batch, but the goal in this algorithm framework is to minimize the processing of multiple microbatches in the iteration, improve the flexibility of resource allocation and lower cloud costs, and improve the resource utilization of traditional server-based model parallel distributed training.

Fu Y et al [9], as the first team to implement large-scale language model migration in a serverless inference system, solves the problem of unpredictable inference delay when dealing with large-scale data models. Cloud computing provides a solid foundation for flexible on-demand provisioning to meet the growing demand for complex services. However, as applications grow in size, the centralized client-server approach used in cloud computing increasingly limits the scalability of applications. But, multi-view detection model (MvDet) [10], a real world multi-view detection model implemented by FastFL C/C++ high-performance edge inference framework, can realize the hyper-scalability of serverless computing.

3. Method

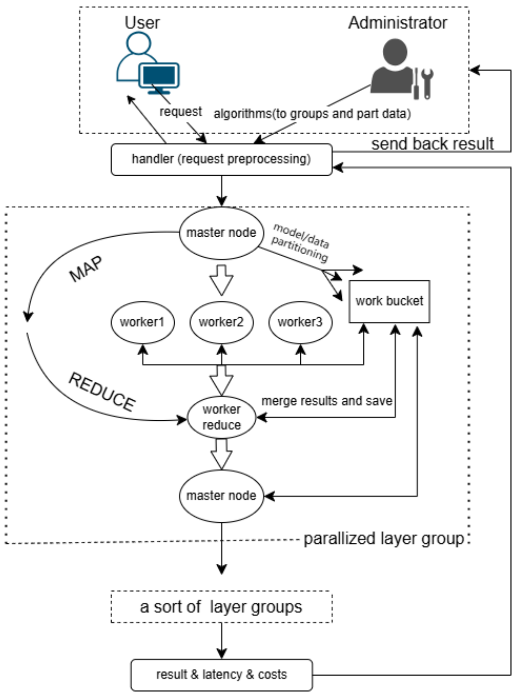


Figure 1. Framework Diagram of Deep Learning Model Inference System Based on Serverless Platform

The paper have designed a distributed inference system for deep learning models based on Aliyun Cloud Function which is a serverless platform. It takes advantage of the elastic and dynamic changes in computing resources and billing methods of the serverless platform, and at the same time overcomes the problems of the limitations on computing resources, including CPU and memory, which results in enormous inference latency.

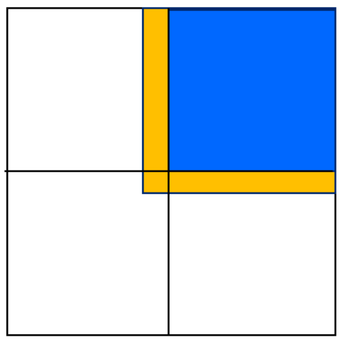
3.1. The Basic Framework

As shown in figure1, the task of the framework is to receive the user's requests for inference, then divide the data according to the pre-designed distributed inference plan, complete the inference task in a parallel way, and finally return the inference result to the user.

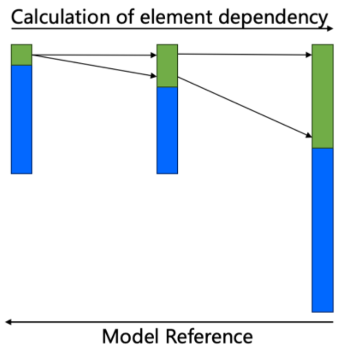
The paper adopt a coarse-grained parallelization strategy to deal with the huge communication costs between the functions on the serverless platform. Specifically, the paper merge some of the element-wise layers that can be parallelized such as convolutional layers and pooling layers into a layer group, and do the data partition and parallelization on layer groups to reduce the total communication cost.

3.1.1. Functions. The service consists of 3 functions that are handler, master, and worker.

3.1.2. Handler. The handler is the entrance of the inference service, and it processes the HTTP requests from users. It provides a public API that allows users to send requests and data. After receiving data, the handler function saves the data into an OSS bucket, then sends a request to the master function through an internal API and starts the main procedure of the inference.



(a). The method of data partition



(b). The bottom-up method for calculating element dependencies

Figure 2. The method of data partition and dependency computation

3.1.3. Master. The master function is the coordinator of the whole inference process and undertakes several key tasks. Firstly, according to the pre-designed parallelization plan based on the specific model, the master function generates a data partition plan for each layer group. Then, it divides the data and forks the corresponding number of worker nodes. The master saves the divided data into the OSS bucket for the workers to read. During the inference procedure of the workers, the master waits for them to complete. After that, the master merges the parts of the results from the workers and goes to the next parallelized or serial step according to the inference plan. For the non-element-wise layers that can't be parallelized, the master would process them directly. Therefore, compared to the workers, the master needs more CPU and memory resources.

3.1.4. Worker. The worker functions are the nodes that do the parallelized inference tasks. After being awakened by the master, a worker would obtain a unique node number and it would get the corresponding data file from the OSS bucket for itself. Then, it does the inference for the data and saves the inference result back to the OSS bucket.

After completing the inference procedure above, the master node returns the inference result to the handler, and the handler returns the result to the user by HTTP response which is the end of the whole inference service procedure.

3.2. Method of data partitioning

As for the method of dividing data for the distributed inference, the former researcher Gillis[1] mentioned 2 possible ways which are dividing by the data's shape and dividing by the data channels. Considering the characteristics of current deep learning platforms such as PyTorch, the paper think dividing by the shape of the data is more realistic. As shown in figure2(a), our method divides the input data equally according to its length and width. Since the parallelized inference involves convolutional layers, it is necessary to consider the dependencies of the output elements on the input elements. To achieve this goal, the paper designed a bottom-up method as shown in figure2(b), which automatically computes the range of the input elements on which the output elements. The algorithm takes the parameters of the convolutional layers, more specifically, the stride, padding, and the shape of the convolution kernel, and makes a mapping from the output elements to the input elements.

4. Results

In this study, a series of experiments were designed and executed with the aim of verifying the rate as well as the cost of different parameters for the frame computation. These experiments provide a better representation of the details of the framework's operation and the overall impact of different resource constraints on the framework's performance. Input random data to simulate the image data and obtain the resource usage required to run as well as the total running time. The experiments are used to obtain practice-based data for improving the design of the framework, as well as to provide detailed reference insights for the subsequent design of patterns close to the application.

The evaluation setup of this study adjusts the batch size, partitioning scheme (worker_num), number of vCPUs, and memory limit to explore the impact on resource utilization, parallel time share, and total runtime in different scenarios. Table1 shows the performance data obtained from monitoring the system under different parameters to explore the processing performance of the system in terms of runtime and memory usage.

Table 1. Performance Data Across Different Parameters

vcpu	memory_size	batch_size	worker_num	max_workertime(ms)	total_memory usage(mb)	total_runtime(ms)	worker/total time
0.35	512	1	72	2814.7	379.24	28639	0.589692378
0.35	512	1	24	2784	389.82	21619	0.772653684
0.35	512	16	72	3069	481.53	50415	0.365248438
0.35	512	16	24	3633.7	481.31	32853	0.663628892
1.5	4096	1	72	2434.2	407.1	21764	0.671071494
1.5	4096	1	24	2292.2	391.8	16724	0.822363071
1.5	4096	16	72	3418	601.6	48218	0.425318346
1.5	4096	16	24	2991.1	567.7	26557	0.675776631
1.5	4096	64	72	4638.9	1242.9	110743	0.251333267
1.5	4096	64	24	4638.5	1142.3	58482	0.475890017
1.5	4096	128	72	6466.7	2020.3	197641	0.196316554

4.1. Total runtime

4.1.1. Impact of the segmentation scheme. In this experiment, two different partitioning schemes are chosen for comparison, namely, the preferred 4*4 partitioning scheme and the fixed 2*2 partitioning scheme, and 72 and 24 worker nodes are generated after 6 layers of layer group, respectively. As shown in figure3(a), the depth of the color represents the length of the total running time, and the heat map shows the relationship between the number of samples as well as the segmentation scheme and the total running time more intuitively, which verifies that the total running time increases with the number of workers and the number of samples under different sample numbers. Figure3(b) quantitatively using bar charts, the total response time of the 24-worker splitting scheme is about 52.6% of that of the 72-worker, and the system exhibits some optimized efficiency as the batch size increases, with the growth rate of the running time decreasing from 121.55% to 78.47%. The increase in the number of partitions led to an increase in the accuracy of the graph processing, but the 72-worker splitting scheme split more nodes leading to more communication costs and greater network latency.

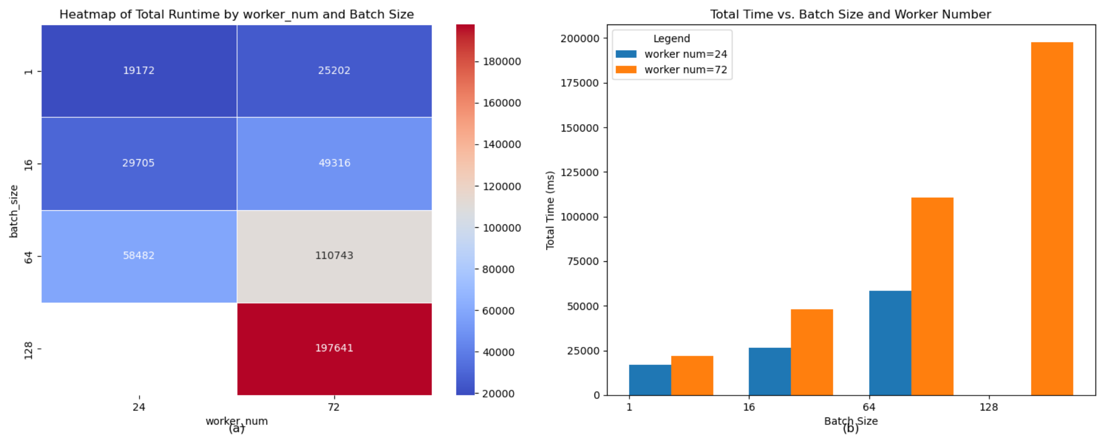


Figure 3. Total Runtime by Sample Number and Worker Count

Parallel technique is used to process the segmented image data, as shown in figure4, which compares the time occupancy of parallel processing under two different segmentation schemes. From the inner circle to the outer circle are the results of different occupancy ratios as the batch size increases, the larger the batch the smaller the parallel occupancy ratio, also proving the superiority of using parallel computing in the case of large batches. Comparing the different segmentation schemes the scheme with more workers has a smaller percentage of parallelism because the data being allocated is more granular and the worker nodes have a shorter processing time, while taking into account the increase in the cost of communication between the nodes, making the parallel time a smaller percentage of the total time.

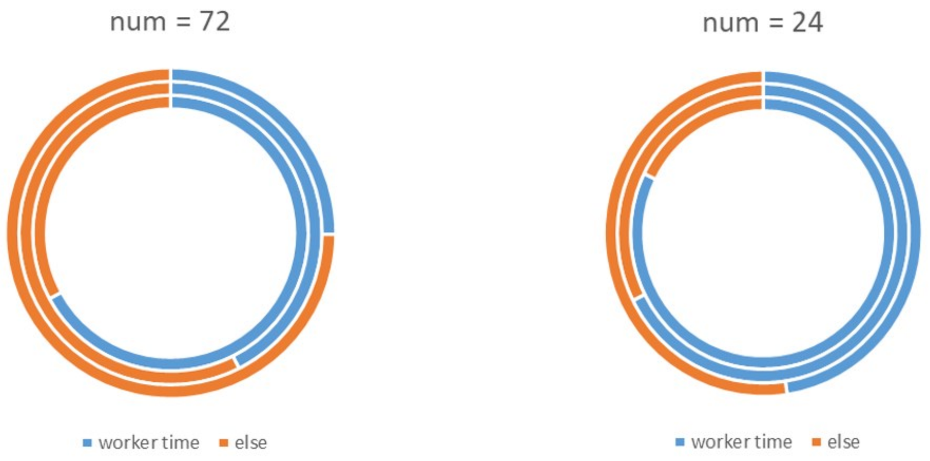


Figure 4. Parallel Processing Time Proportion by Batch Size

4.1.2. Effect of number of vCPUs and memory size. In the test, the number of vCPUs and memory size are intrinsically related to each other considering the principle of resource balancing. In order to comprehensively evaluate the impact of these two parameters on the system performance, the paper have adjusted the number of vCPUs and memory size at the same time to more accurately simulate the application scenarios under different hardware configurations. As shown in figure5, the depth of the color represents the length of the total running time, the heatmap visualizes the relationship between the number of vCPUs and the total running time of the memory size, the total running time decreases as the number of vCPUs and the size of the memory increases, increasing the number of vCPUs improves the rate of computation for simultaneous multitasking, resulting in a reduction in the speed of parallel computation, and a significant reduction of the time required to execute a multithreaded program.

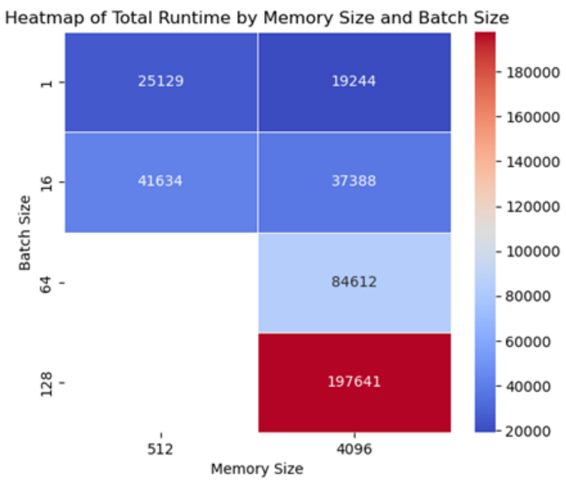


Figure 5. Total Runtime by vCPU Count and Memory Size

The bar chart in figure6 shows the percentage of parallel time at different batches as well as the number of vCPUs and memory size, and the comparison reveals that the percentage of parallel time with more vCPUs is greater than the data with fewer vCPUs. However, this does not imply an increase in parallel time; this phenomenon occurs because the increase in vCPUs reduces latency and throughput, and for the elevated utilization of resources, the change in total runtime is close to 30%, and the decrease in parallel rate is relatively flat, leading to a situation where the percentage of parallel time rises instead.

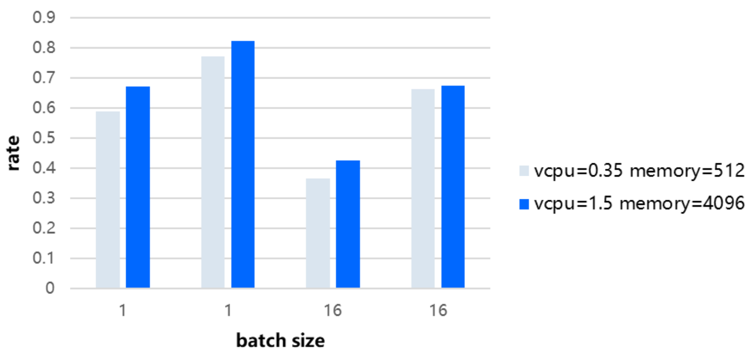


Figure 6. Parallel Time Proportion by Batch Size and vCPU Count

4.2. *Memory usage*

Controlling the above two variables, the paper observe their effects on the total memory usage, and the experiment can provide important data for cost control as well as performance optimization. As shown in figure7(a), the heatmap visually illustrates that an increase in the number of workers and the number of samples likewise leads to an increase in the memory footprint, with each worker requiring a certain amount of memory to store the program code, maintain the run state, and process the data. Figure7(b) shows the impact of VCPU and memorysize on memory usage - VCPU also takes up more memory, given the faster operation.

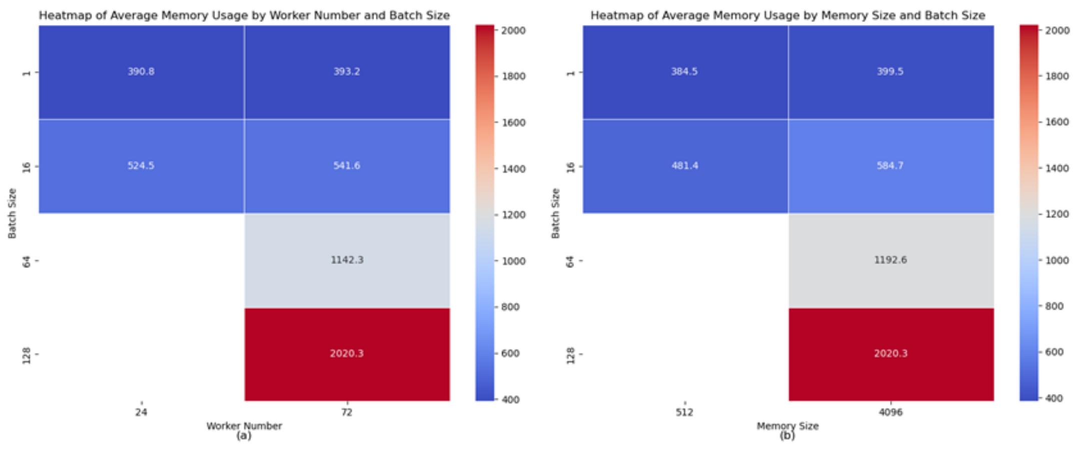


Figure 7. Memory Usage by Different Parameters

5. Conclusion

In this research on using serverless platforms to realize parallelized deep learning model inference, the paper have proposed a feasible approach and designed a practical framework and an algorithm for deploying and accomplishing distributed deep learning model inference tasks on Aliyun's serverless computing service. The paper have tested our framework and algorithm on the ResNet-18 model and analyzed the testing results. The paper have discovered that it is possible to use a small amount of memory and CPU resources to infer the model on medium-sized data, while the time consumed in the parallel inference part can be achieved to account for about 3/4 of the total runtime. Compared to the traditional serial inference calculation in one function, the paper could effectively make the model inference which costs a huge amount of memory be completed in a relatively short time in the serverless functions whose resources are strictly limited.

However, on the other hand, each worker's memory requirement could still reach 4GB in the inference of large data, which causes the total memory required in all nodes is still too enormous compared to the general serial methods. At the same time, the use of HTTP requests and OSS objects for data storage in the communication between the functions also results in excessive communication costs in the case of high parallelism, which makes communication the bottleneck of the whole procedure. This may counteract the optimization of the computation time, making it difficult to apply this framework to real-world services which have limitations on total resource consumption and high requirements on time cost.

In future works, one could work on adjusting the framework to decrease the resource requirement of the functions by such as model compression and dynamic resource allocation strategy which dynamically adjust the resource for each node according to the characteristics

Authors Contribution

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

References

[1] Gao, W., Hu, Q., Ye, Z. and et al. 2022. Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision. arXiv preprint, arXiv:2205.11913.
[2] Zhang, C., Xia, J., Yang, B. and et al. 2021. Citadel: Protecting data privacy and model confidentiality for collaborative learning. Proceedings of the ACM Symposium on Cloud Computing, 546-561.
[3] Yu, H., Li, J., Hua, Y. and et al. 2024. Cheaper and Faster: Distributed Deep Reinforcement Learning with Serverless Computing.

[4] Pei, Q., Yuan, Y., Hu, H. and et al. 2023. AsyFunc: A high-performance and resource-efficient serverless inference system via asymmetric functions. Proceedings of the 2023 ACM Symposium on Cloud Computing. 324-340.

[5] Chahal, D., Palepu, S., Mishra, M. and et al. 2020. SLA-aware workload scheduling using hybrid cloud services. Proceedings of the 1st Workshop on High Performance Serverless Computing. 1-4.

[6] Oakley, J. and Ferhatosmanoglu, H. 2024. FSD-Inference: Fully Serverless Distributed Inference with Scalable Cloud Communication. arXiv preprint arXiv:2403.15195.

[7] Yu, Minchen and et al. 2021. "Gillis: Serving large neural networks in serverless functions with automatic model partitioning." 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS).

[8] Liu, Y., Jiang, B., Guo, T. and et al. 2022. Funcpipe: A pipelined serverless framework for fast and cost-efficient training of deep learning models. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 6(3): 1-30.

[9] Fu, Y., Xue, L., Huang, Y. and et al. 2024. ServerlessLLM: Locality-Enhanced Serverless Inference for Large Language Models. arXiv preprint arXiv:2401.14351.

[10] Mittone, G., Malenza, G., Aldinucci, M. and et al. 2023. Distributed Edge Inference: an Experimental Study on Multiview Detection. Proceedings of the IEEE. ACM 16th International Conference on Utility and Cloud Computing. 1-6.