Second-hand Device Price Evaluation System:

Serverless Deployment of Machine Learning Model

Group 6: LI, Yifan; YIP, Valerie Fang Hong; HUANG, Yifu; HUANG, Ziyan

1. Introduction

The development of applications has undergone a significant transformation in recent years with the advent of serverless architecture. Traditionally, deploying software required managing physical or virtual servers with overhead such as infrastructure provisioning, maintenance, and scaling. However, with the rise of serverless architecture, the infrastructure is abstracted away which allows developers to focus on writing the business logic. Frameworks such as OpenFaaS and Kubeless have exemplified this shift.

This report will specifically explore the deployment of a machine learning model using a serverless architecture, comparing its advantages and challenges to traditional server-based methods. Key aspects such as performance, cost, and scalability will be analyzed to provide a comprehensive understanding of how these two deployment strategies differ. Through this analysis, we aim to highlight the potential of serverless architecture and make recommendations for cases where serverless makes an excellent choice.

2. Background

For this project, we chose to deploy a neural network machine learning model using OpenFaaS within a Kubernetes environment, specifically deploying it on Amazon Elastic Kubernetes Service (EKS).

The machine learning model takes as input various device specifications and predicts the used price (after depreciation), normalized based on factors like the device's condition, age, and features. The prediction would reflect how much the device is worth after being used for a certain number of days (days_used), along with the other features like brand, screen size, RAM, etc.

3. Server-Based Architecture

In the traditional server-based architecture, developers manage physical or virtual servers, handling everything from server provisioning, maintenance of application, and load balancing. This approach offers control over the hardware but it can be resource-intensive.

There are on-premises servers, where the physical hardware servers needed are maintained by companies in house. In contrast, there are cloud servers, where cloud providers like AWS maintain the physical hardware while making virtual or bare metal servers available to customers (IBM, n.d.). Both on-premises and cloud servers can be used in a server-based architecture.

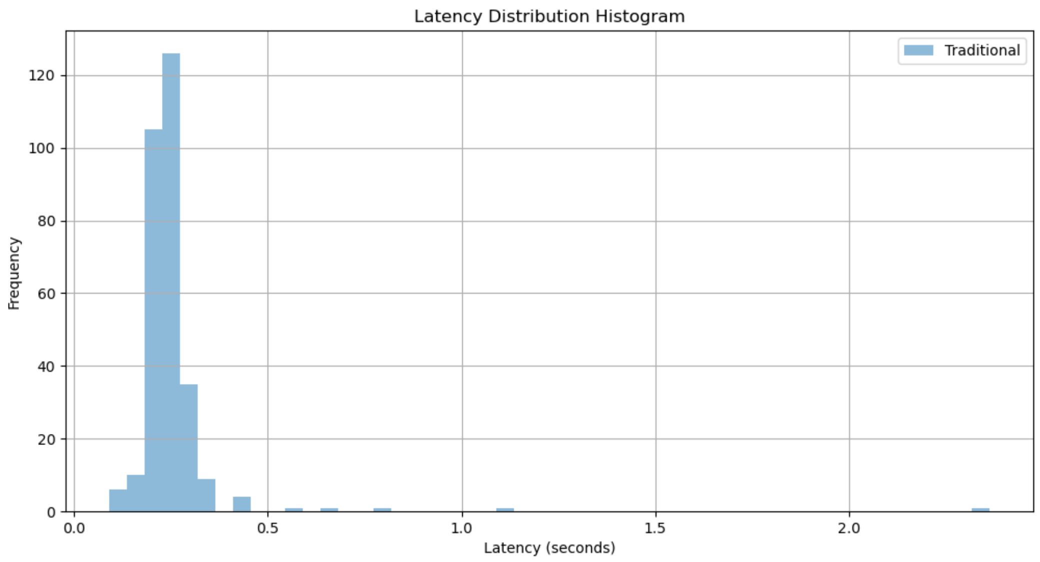
4. Serverless Architecture

A serverless architecture is a way to build and run applications and services without having to manage infrastructure. (AWS, 2024). The underlying infrastructure is abstracted away, allowing developers to focus on writing application code and the business logic. In this architecture, applications are broken down into functions that are executed in response to events. For example, in OpenFaaS, the platform that we have chosen, functions can be run on-demand, or on a schedule, and can be triggered by events from your existing systems like Apache Kafka or AWS. (OpenFaaS, n.d.).

5. Implementation and Comparison

5.1. Performance

We implemented a traditional server-based application with Flask and tested its latency locally



The performance test results indicate that the Flask application is functioning efficiently under the current load of 100 requests, with all requests successfully processed and an error rate of 0.00%. **The average latency** is 0.2439 seconds, which is acceptable for most web applications, though the maximum latency of 1.2478 seconds suggests occasional spikes that could be optimized. The application achieved a **throughput** of 39.65 requests per second, completing all requests in 2.64 seconds. While the results are promising, further testing under higher loads and optimizations for latency spikes and scalability (e.g., using a production-grade WSGI server like Gunicorn) will ensure reliable performance as traffic increases.

One of the key performance considerations for OpenFaaS deployment on EKS Cluster is cold start latency. When a function hasn't been invoked for a while, Kubernetes may scale down the corresponding pods to save resources. The next invocation would require the pod to be restarted, causing a delay. This cold start latency depends on factors such as:

- The size of the container image. Since our model needs a Tensorflow library as inference runtime, the docker image constructed is relatively high. When using the official image from Tensorflow as base image, the total size is up to more than 3 GB. Which is huge if we scale it up.
- EKS cluster configuration and resource availability. Some non-optimal configurations will lower the starting time.

For latency-sensitive applications, keeping some pods "warm" (always running) can mitigate this issue, though it increases resource usage and costs. It is believed that in a cold start mode, it takes around 2 seconds for the serverless-deployed model to react, which is significantly slower than the one deployed locally.

5.2. Scalability

5.2.1. Advantages of Serverless

a. Automatic Scaling for model workloads

1). Scalability in model deployments is crucial, especially when models are exposed to varying workloads. Serverless architectures (like OpenFaaS) offer automatic scaling, meaning that when the model is deployed and receives an increased number of inference requests, it can automatically spin up new instances of the model without manual intervention. **2)**. The inference demand for the models (e.g., depreciation_model.h5) may vary, and the serverless architecture ensures that the system can handle unpredictable loads efficiently without having to manually provision additional resources.

b. Efficient Resource Utilization

Since machine learning inference often involves variable resource usage (e.g., depending on the complexity of the model or the volume of data), serverless architectures can dynamically allocate resources. This helps in maintaining cost-efficiency by only charging for actual compute time, which is beneficial in applications that may not have continuous high-demand.

5.2.2. Disadvantages of Serverless

a. Cold Starts Impact on Latency

1). For real-time inference in applications, a cold start in a serverless function (i.e., when an idle function is invoked) can introduce latency. This is especially important if the model needs to provide quick responses (e.g., in medical diagnostics or real-time recommendation systems). **2)**. If the model is used in such contexts, cold starts could affect the performance and user experience, as functions may take longer to initialize before processing the request.

b.Resource Limitations for Heavy Models:

1). Some models (especially deep learning models or large models) require significant resources such as memory and computational power. Serverless platforms often have resource limits (e.g., AWS Lambda limits memory and execution time), which can restrict the performance of heavy models. **2)**. If the deployed model (e.g., depreciation_model.h5) requires substantial computational resources or long inference times, the serverless environment may not be ideal for such workloads. In this case, traditional server-based deployments (e.g., deploying the model on a dedicated server or Kubernetes pods with specific resource allocations) may perform better.

5.3. Deployability

5.3.1. Advantages of Serverless

a. Quick Deployments and Updates

1). Serverless architectures allow for rapid deployment and updates of the model. Once the model is trained (using train.py), it can be quickly packaged and deployed using faas-cli to OpenFaaS, enabling immediate updates to the model without needing to reconfigure the backend. **2)**. This is advantageous in a project where you may want to quickly iterate on the model based on new data or test different model configurations (e.g., training new versions of the model).

b.Simplified Backend Configuration:

1). Serverless platforms abstract away the complexity of backend infrastructure, making it easier to focus on model development and function logic. For example, by using OpenFaaS on Kubernetes, you avoid the need to manually configure servers, load balancers, or manage scaling. **2)**. This makes the process of deploying models much simpler and quicker, especially for proof-of-concept deployments where infrastructure management is not a priority.

5.3.2. Disadvantages of Serverless

a.Testing and Debugging Challenges:

1). Once deployed to a serverless environment, testing and debugging become more challenging due to lack of visibility into the backend infrastructure. **2)**. This can lead to difficulties when troubleshooting issues in the deployed model (e.g., incorrect predictions or errors in model inference). Since traditional debugging tools might not be available in serverless environments, you may need to rely on specialized monitoring and logging tools (e.g., AWS CloudWatch) to understand the model's behavior.

b.Not Suitable for Long-Running Processes:
1). Many tasks require longer execution times, especially when models are performing heavy computations or when inference requires processing large datasets. However, serverless functions typically have execution time limits (e.g., AWS Lambda's time limit is 15 minutes), which may not suit applications involving long-running processes or batch jobs. **2)**. If the model requires extended time for batch inference or processing of large datasets, serverless architecture may not be suitable. In such cases, using traditional server-based or containerized solutions (e.g., deploying the model as a Kubernetes pod) may be more appropriate.

6. Conclusion

In this project, we built a machine learning model with neural network algorithm for the second-hand device price evaluation, and deployed the model using OpenFaaS in a Kubernetes environment. To validate the difference between two employment modes, we test the response performance in both server-based architecture and serverless architecture, with metrics of response performance including the time for completing 100 requests, average latency, error rate and throughput. It turned out that the model deployed in a serverless architecture has a significantly slower response rate, due to the cold start effect. Furthermore, we also comprehensively discussed the difference between server-based deployment and serverless deployment in the aspects of scalability and deployability, which objectively provides more insights into the advantages and disadvantages of two architectures.

7. References

- AWS, 2024. *Building Applications with Serverless Architectures.* Retrieved from https://aws.amazon.com/lambda/serverless-architectures-learn-more/, 18 November 2024.
- IBM, n.d. *What is a cloud server?* Retrieved from https://www.ibm.com/topics/cloud-server, 18 November 2024.
- OpenFaaS, n.d. *Serverless Functions, Made Simple.* Retrieved from https://www.openfaas.com/, 18 November 2024.