ANM 2023 Spring Project (Phase 1)

Anomaly Localization in Microservices System

Background

As the Internet has gradually become more popular and people's reliance on Internet applications has increased, the technology for developing Internet applications is changing rapidly. In traditional Internet applications, the back-end (server-side) application development usually uses the monolithic program model, meaning that a single program must implement almost all the back-end software functions. However, with the rapid expansion of functional requirements and rapid changes in application traffic, this monolithic program development approach is too large and inflexible for rapid new feature development and elastic scaling of application services, which poses significant challenges for software development and maintenance.

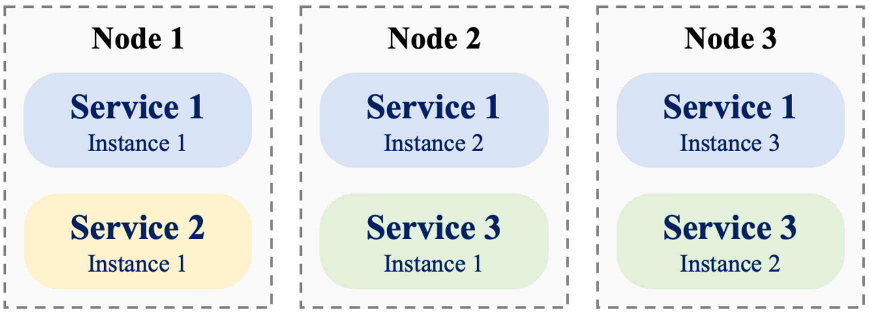

Fig. 1 An example of node and microservices.

Therefore, in today's back-end software development, Internet applications are often split into many microservices, each responsible for a part of the business functions. The microservices call each other through the network to communicate. Since the functions of each microservice are relatively independent, different microservices can be developed and managed by different teams. Another significant advantage of microservices is their scalability. In today's microservice deployment and maintenance process, containers are usually used to realize the operation of services. Each container usually contains one instance of microservice inside, and each physical server (node) can run multiple containers to fully utilize physical server resources. To cope with rapid changes in traffic, each instance of a microservice can have multiple replicas, one running through a separate container, each usually with the same functionality. When the microservice system is running, traffic is evenly distributed to different replicas of the microservice running through the system's internal load balancing (LB) to cope with changes in access traffic. The operators can realize the management of container expansion and rolling updates through container orchestration tools such as Kubernetes[1].

More concepts about microservices and Kubernetes can be found at https://kubernetes.io/docs/concepts/.

[1] https://kubernetes.io/

Task

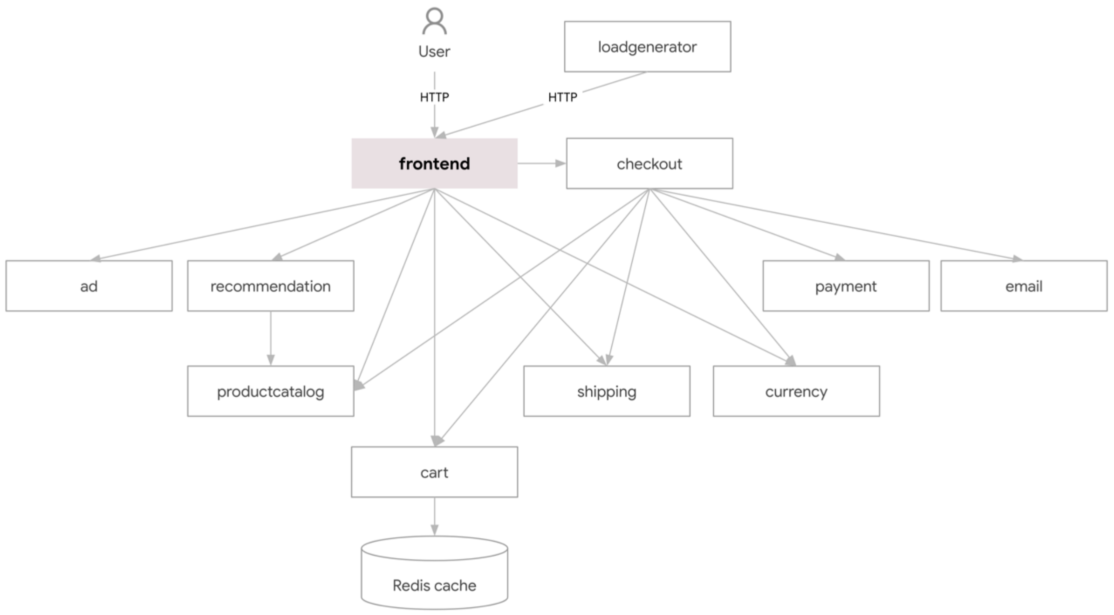

Fig. 2 Architecture of the microservice system in this Project.

Hipster shop is a microservice system, as shown in the figure above. Each microservice may have multiple replicas (containers), and each container may run on a different server node. Now we have several failure cases of this microservice system. One of the containers in this microservice system fails in each failure case, causing fluctuations. Now, please **design and implement an algorithm** to locate the root cause container that caused the failure in each case.

Dataset

In this project, we will provide you with processed metrics data for each container. There are two folders in this dataset: *train* and *test*. The training dataset contains no failures. The testing dataset contains many failure cases under the *processed* folder. Each failure case is represented by an integer number, as shown below. The folder corresponding to each failure contains the metrics data of each container (e.g., *adservice-1* denotes a container of *adservice* with id 1) between the 15 minutes before and the 15 minutes after the failure time point. We also provide you with the *label.json*, which contains the list of failure timestamps and the root cause containers.

```
├── test
│    ├── label.json
│    └── processed
│          ├── 0
│          │    ├── adservice-0.csv
│          │    ├── adservice-1.csv
│          │    ├── …….
│          ├── 1
│          │    ├── adservice-0.csv
│        │     ├── adservice-1.csv
│        │     ├── …….
│        ├── ……
└── train
    └── processed
        ├── adservice-0.csv
        ├── adservice-1.csv
```

For ease of understanding, we will provide a **sample code** for you. This code contains data reading, evaluation, and a very simple root cause localization algorithm. Please note that you can write your own code and do not have to modify it based on this code.

Submission

Phase 1

Start: Apr. 17th (Week 9)
Due: May 15th (Week 13)
Description: Form a team of 2-3 people to complete the task. We will provide training, test data, and labels for the first phase. **Each team** should **submit a report in pdf format** detailing the design of your algorithm and its effectiveness. This report has no length requirement but should include details of your algorithm design and data analysis. This report will account for **20%** of your total score. All team members will be equally scored.
Submission: A report in pdf format (on web-learning).

Phase 2 (TBD)

Start: May 16th (Week 13)
Due: June 5th (Week 16)
Description: We will provide an online judge platform and use additional data to evaluate each team's code. We will provide you with the Phase 2 data before Phase 2 begins. However, we will not provide labels for Phase 2. Therefore, you will need to improve the design of your algorithm to ensure its performance. This online judge platform has a limit on the number of submissions (5 per day per team). We will provide you with detailed submission instructions and a sample code before the start of Phase 2. In the final class (Week 16), each team will be required to give a presentation on your algorithm design and improvement. The score for Phase 2 will consist of **the leaderboard** score and the **presentation score**. You will be provided with specific requirements and a detailed scoring method at the beginning of Phase 2. This part will account for 45% of your total score.
Submission: Source code (on the online judge platform). Final source code (after the presentation, on web-learning). Presentation. Presentation slides (after the presentation, on web-learning).