COSC3P95A2Outline

COSC 3P95- Software Analysis & Testing

Assignment 2

Due date: Sunday, Nov 19th, 2023, at **23:59** (11:59 pm)

Attention:

- This is a **group** assignment designed for a maximum of two students. You have the option to complete the assignment either individually or as part of a two-person group.
- Combine all textual and visual elements into a single PDF file. This includes explaining code snippets, screenshots, and your report. Make sure this PDF is well-organized and easy to navigate.
- You may also include a zip file containing the complete source code, Open-Telemetry configurations, and any other relevant files.
    - Alternatively, you can provide a link to a GitHub repository that contains all these elements. If you choose this method, ensure you have met the criteria outlined in the "Extra Credit Assignment."
- Each group should submit only one set of files. Make sure to clearly indicate the names and student IDs of all group members in your submission.
- In the introduction of your assignment, provide a detailed explanation outlining the contributions made by each group member. Specify what parts of the assignments each person worked on and the extent of their involvement.
    - Failure to include this section may result in a 15-point reduction for each group member.
- This assignment has 100 + 10 (bonus) points and is worth 10% of the course grade. Please also check the Late Assignment Policy.

Name1: DI WU Student ID 1: 6345912

Name2: Daniel Yang Student ID 2: 6937601

Di Wu is responsible for Responsible for implementing automated and manual instrumentation using Open-Telemetry, collecting relevant data or metrics using Open-Telemetry and using Open-Telemetry compliant external tools for visualizing span and

telemetry data in the first part.

Daniel Yang is responsible for the local folder should contain at least 20 files of different sizes, from 5KB to 100MB, implementing client and server applications, implementing and analyzing at least two different sampling strategies: AlwaysOn and Probabilistic Sampling and Selective and Union Implement three advanced features in the first part.

Both of us are responsible for evaluating how each selected feature or their combination affects the system in terms of latency (time), throughput (speed), error rate, or other factors and writing reports.

Both of us are responsible for the part 2 and we run the code to collect enough data. Then together we analyze the data and determine the root cause of the error.

In the end, Di Wu is responsible for creating a GitHub repository

Report

Q1

The application is structured as a client component. The server side is written in Python, listens for client connections through sockets, and receives files from the client. The server uses OpenTelemetry to record trace data. At the same time, the client is also written in Python, connects to the server through a socket, and sends local files to the server. The client also uses OpenTelemetry to record trace data.

Implementation details:
opentelemetry-sdk: OpenTelemetry's SDK library for creating tracking instances and setting up exporters.
opentelemetry-exporter-jaeger-thrift: Exporter for exporting trace data to Jaeger.

First, we use OpenTelemetry for custom probability sampling. Both the client and server are configured with OpenTelemetry to track network operations. Implemented the CustomProbabilitySampler class, allowing for a configurable tracking rate of 40%, thus enhancing traceability without excessive storage due to tracking data. Secondly, we are equipped with try-except blocks on both the client and the server to handle exceptions, especially exceptions caused by socket operations. This ensures that the application does not crash unexpectedly and provides meaningful error messages for debugging. Not only that, the client attempts to establish a connection to the server with the specified number of retries (RETRY_LIMIT) and interval (RETRY_INTERVAL). This mechanism enhances client reliability under unstable network conditions. Finally, the client reads the files from the specified directory and sends them to the server over the socket connection. The server receives these files and saves them locally.

Performance Analysis:
Sampling strategy: CustomProbabilitySampler effectively reduces the amount of tracking data, which is beneficial to performance, especially in high-throughput scenarios. It strikes a balance between observability and resource usage.

Network communication: Retry logic in the client increases application resiliency. However, under unstable network conditions this may cause delays. The current implementation does not parallelize file transfers, which can become a bottleneck in scenarios that require high throughput.

Error handling: Applications become more robust to network-related exceptions. Properly logging errors helps with troubleshooting.

Retry mechanism: This feature has proven to be crucial for maintaining client functionality in the event of temporary network failures or server unavailability.

Challenges faced：

Network Stability: Implementing a retry mechanism that balances excessive fast retries with long delays is challenging. This requires careful adjustment based on expected network conditions.
Telemetry data volume: Balancing the granularity of tracking data and the overhead it introduces is a challenge. Too much data can slow down applications and increase costs, while too little data can hinder observability.
Concurrency: Efficiently handling multiple client connections on the server side requires the implementation of threads. Ensuring thread safety and managing resources efficiently is critical.

Q2

We intentionally introduced a bug where the client would have a 30% chance of introducing a delay when sending files, and this delay would increase the transfer time.

First, the predicate we can determine is that the file transfer request will be received on the server side, given the delay introduced by the client. Because we know that under normal circumstances there will be no delays. We then run the client-server application multiple times and log the data for each execution. We recorded the duration of each file transfer and whether delays were introduced.
Then, we calculated two statistical indicators, Failure Metrics and Increase Metrics. Failure Metrics are situations where file transfer duration increases significantly when latency is introduced. We identify failures by comparing the duration of each transfer while Increase Metrics are situations where the duration of a file transfer increases significantly relative to normal conditions when latency is introduced. We calculated the average of the duration and compared it with the average under normal conditions. Analyzing the statistical metrics, we found that the average value of duration increases significantly when delay is introduced. This indicates that latency is the root cause of increased transmission times. To solve this problem, we checked the client code, found the part that introduced the delay, and fixed it to normal transmission without introducing delay. Afterwards, we ran the client-server application again to ensure that the fixed code no longer introduced delays and that file transfers worked correctly in all cases. Finally, we successfully identified the root cause of the introduced error and fixed the code.

GitHub repository:
https://github.com/Asukadd/COSC3P95A2.git