COSC 3P95 Assignment 2 Report

Nicholas Maspoch (7938335)

Daniel Maspoch (7913379)

GitHub Repository (contains README)

Part 1 Report

Introduction

Nicholas focused on the coding portion of the assignment, using various sources (provided below in the implementation section) to develop the client-server program, setup instrumentation, implement sampling and advanced features, and export data to OpenTelemetryCollector and visualize it using Jaeger and Prometheus.

Daniel focused on the analytical portion of the assignment, writing this report and describing the findings from the visualization and analysis tools. He also created the tables summarizing the results.

As mentioned above, Nicholas compiled the various sources he used to implement the application and set up the visualization and analysis tools. Claude.ai was used to address any potential issues in the code and to inject the bug for the statistical debugging portion of the assignment. It was also used to generate test files (test.py and analyze_traces.py) with the former being used for part 1 and the latter being used to analyze JSON files exported from Jaeger containing traces for the client and server respectively and calculate latency, throughput, and more. Finally, it was used to analyze the results from analyze_traces.py to identify any patterns to mention for this report.

Implementation Details

This assignment was implemented using the Python programming language and these sources:

- Client-server file transfer system
    - Additional source
- Get directory path
- Create file of a particular size
- Delete files in directory
- ThreadPoolExecutor
- Sampling
- Compression using zlib library
- Encryption using Fernet library
- Setting up OpenTelemetryCollector, Jaeger, and Prometheus Docker containers

- The OpenTelemetry documentation was used to set up instrumentation

Instrumentation Design

For the client, three spans were created:

- The file_generation_span recorded the number of files generated and logged when each was created.
- The client_span tracked the total number of bytes generated, the total number of files created, and the time it took to do so, with logs for when each file was sent.
- The sent_file_span contained the original size and sizes after compression and then encryption respectively. It also recorded whether the file was compressed, as sometimes the compression method failed to compress the size, and, if so, the compression ratio. It would be logged when the file was compressed, encrypted, and uploaded to the server.

These metrics aided in the calculations used for latency, throughput, and more.

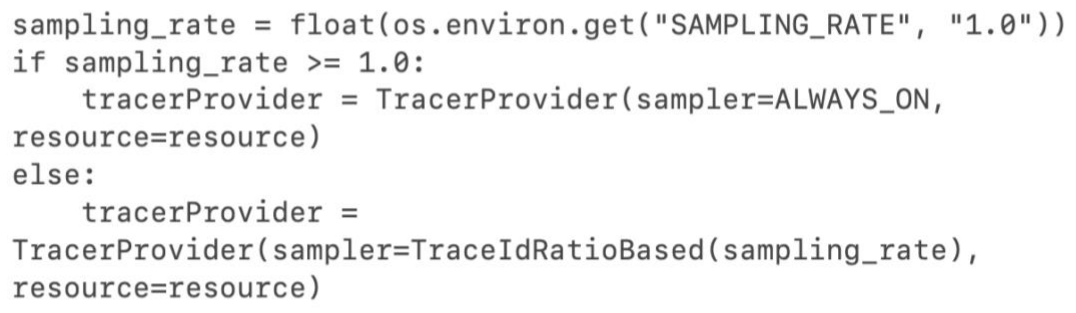
For the server, two spans were created:

- The file_span shared the same attributes used for the sent_file_span used for the client. The number of chunks received was also recorded. These were used to verify that data was correctly sent from client to server.
- The client_span was used for logging purposes, such as outputting the number of files expected and when each file was received.

Experimental Results and Analysis

This analysis implements two OpenTelemetry sampling strategies, AlwaysOn and 25% Probability Sampling, in a file transfer client-server system. Comparing the results reveal surprising performance characteristics and important trade-offs between observability coverage and system behavior.

Both client and server use TraceIdRatioBased Sampling:



```
sampling_rate = float(os.environ.get("SAMPLING_RATE", "1.0"))
if sampling_rate >= 1.0:
    tracerProvider = TracerProvider(sampler=ALWAYS_ON,
resource=resource)
else:
    tracerProvider =
TracerProvider(sampler=TraceIdRatioBased(sampling_rate),
resource=resource)
```

With this sampling, when a client trace is sampled, all related server spans are also sampled, ensuring consistency across all traces. Furthermore, parent-child span relationships are maintained.

We use the AlwaysOn sampling method (sampling_rate = 1.0) to capture 100% of traces. This method is applied to both client.py and server.py files.

We also use the 25% probability sampling method (sampling_rate = 0.25) to capture about a quarter of the traces. Once again, this method is applied to both client.py and server.py files.

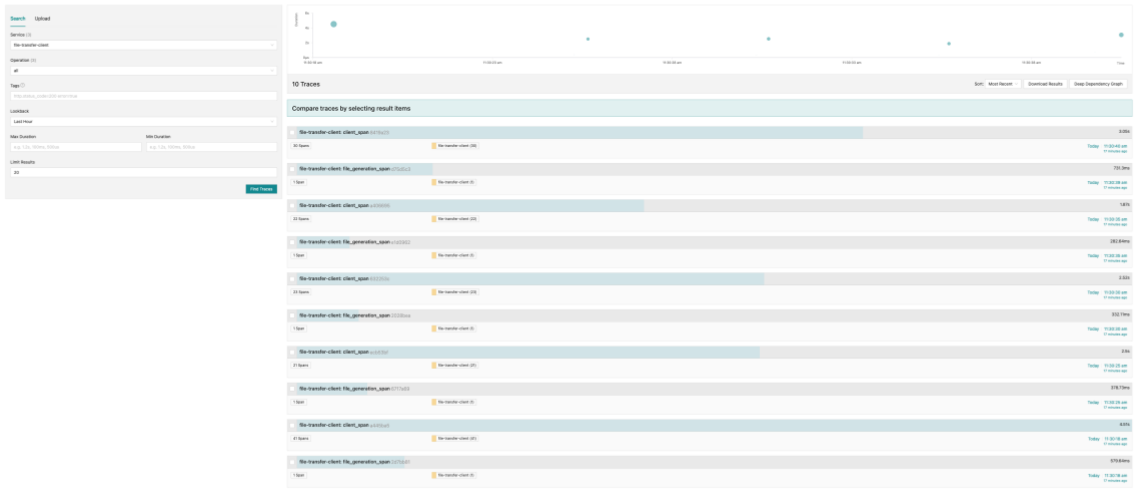
Both files are using the same sampling rates, so we maintain trace continuity across the distributed system.

We analyze 5 runs for each configuration (i.e. 20 runs executed with a sampling rate of 25 %). Files ranging from 5KB to 100MB are randomly generated with compression and encryption.

Jaeger and Prometheus Screenshots

File-Transfer-Client

Client Trace Results

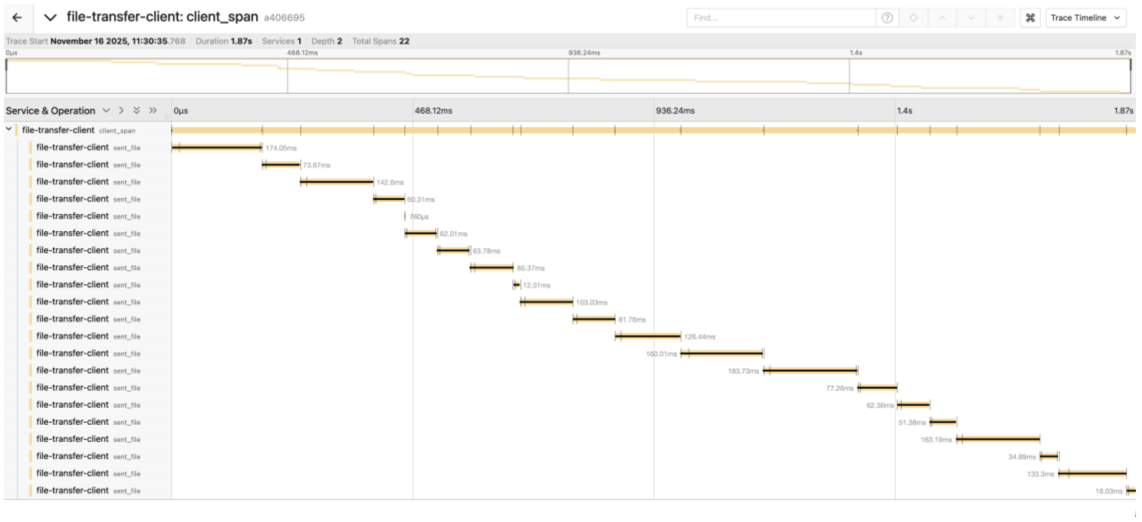


10 Traces

Compare traces by selecting result items

file-transfer-client: client_span — 3.05s
file-transfer-client: file_generation_span — 731.3ms
file-transfer-client: client_span — 1.87s
file-transfer-client: file_generation_span — 282.84ms
file-transfer-client: client_span — 2.53s
file-transfer-client: file_generation_span — 332.71ms
file-transfer-client: client_span — 2.6s
file-transfer-client: file_generation_span — 378.73ms
file-transfer-client: client_span
file-transfer-client: file_generation_span

File Transfer Client Trace View

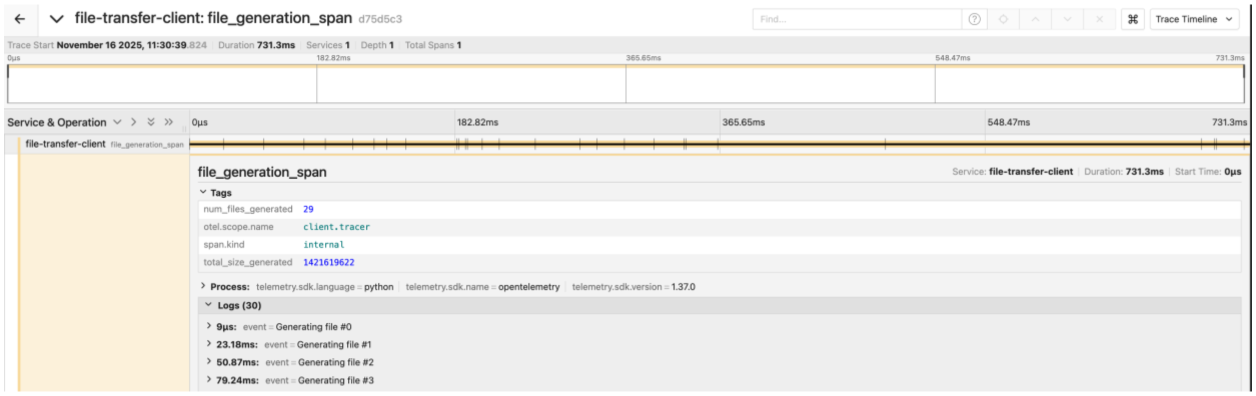


file-transfer-client: client_span a406695

Trace Start November 16 2025, 11:30:35.768 | Duration 1.87s | Services 1 | Depth 2 | Total Spans 22

```
Service & Operation                    0µs   468.12ms   936.24ms   1.4s   1.87s
file-transfer-client client_span
  file-transfer-client sent_file   174.05ms
  file-transfer-client sent_file   73.67ms
  file-transfer-client sent_file   142.6ms
  file-transfer-client sent_file   60.31ms
  file-transfer-client sent_file   760µs
  file-transfer-client sent_file   62.01ms
  file-transfer-client sent_file   63.78ms
  file-transfer-client sent_file   85.37ms
  file-transfer-client sent_file   12.01ms
  file-transfer-client sent_file   103.03ms
  file-transfer-client sent_file   81.76ms
  file-transfer-client sent_file   126.44ms
  file-transfer-client sent_file   160.01ms
  file-transfer-client sent_file   183.73ms
  file-transfer-client sent_file   77.26ms
  file-transfer-client sent_file   62.36ms
  file-transfer-client sent_file   51.38ms
  file-transfer-client sent_file   163.19ms
  file-transfer-client sent_file   34.89ms
  file-transfer-client sent_file   133.3ms
  file-transfer-client sent_file   18.03ms
```

File Generation Span



file-transfer-client: file_generation_span d75d5c3

Trace Start **November 16 2025, 11:30:39**.824 | Duration **731.3ms** | Services **1** | Depth **1** | Total Spans **1**

Service & Operation | 0µs | 182.82ms | 365.65ms | 548.47ms | 731.3ms

file-transfer-client file_generation_span

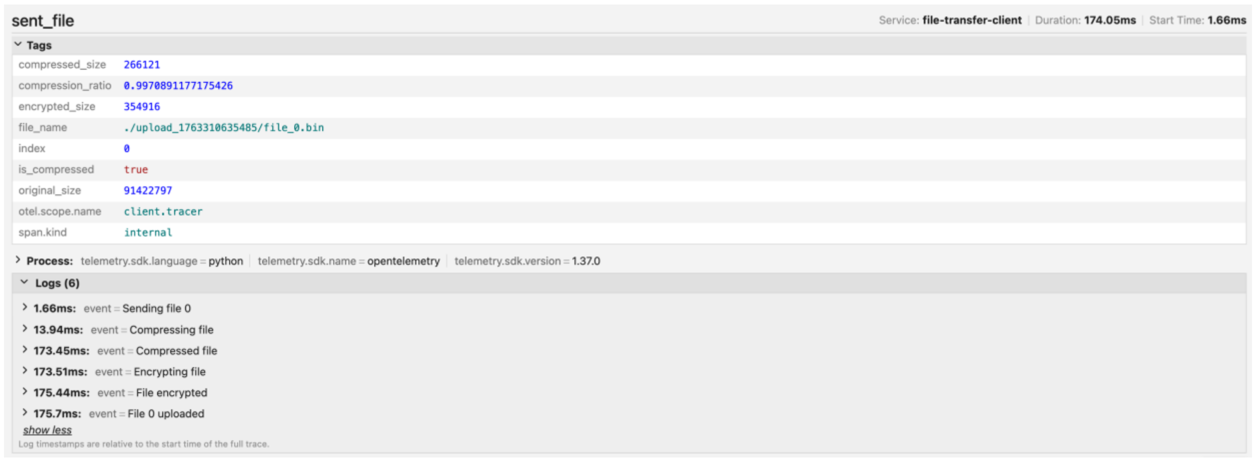
file_generation_span — Service: **file-transfer-client** | Duration: **731.3ms** | Start Time: **0µs**

Tags

num_files_generated	29
otel.scope.name	client.tracer
span.kind	internal
total_size_generated	1421619622

```
> Process: telemetry.sdk.language = python | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.37.0
v Logs (30)
> 9µs: event = Generating file #0
> 23.18ms: event = Generating file #1
> 50.87ms: event = Generating file #2
> 79.24ms: event = Generating file #3
```

Sent File Span



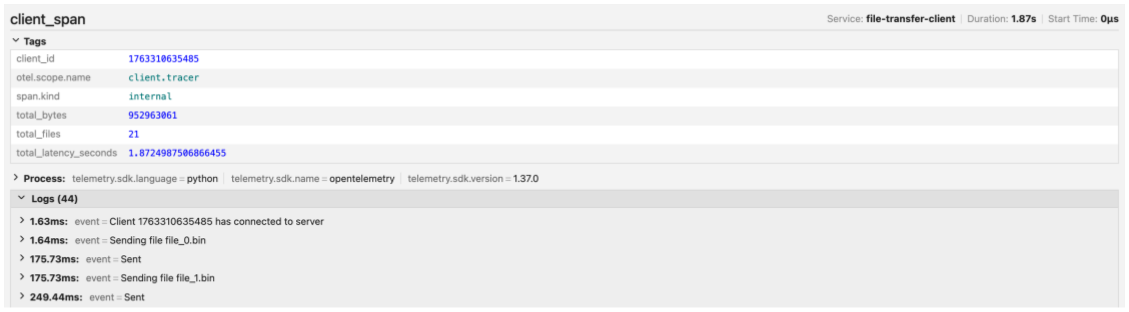
sent_file — Service: **file-transfer-client** | Duration: **174.05ms** | Start Time: **1.66ms**

Tags

compressed_size	266121
compression_ratio	0.9970891177175426
encrypted_size	354916
file_name	./upload_1763310635485/file_0.bin
index	0
is_compressed	true
original_size	91422797
otel.scope.name	client.tracer
span.kind	internal

```
> Process: telemetry.sdk.language = python | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.37.0
v Logs (6)
> 1.66ms: event = Sending file 0
> 13.94ms: event = Compressing file
> 173.45ms: event = Compressed file
> 173.51ms: event = Encrypting file
> 175.44ms: event = File encrypted
> 175.7ms: event = File 0 uploaded
show less
Log timestamps are relative to the start time of the full trace.
```

Client Span



client_span — Service: **file-transfer-client** | Duration: **1.87s** | Start Time: **0µs**

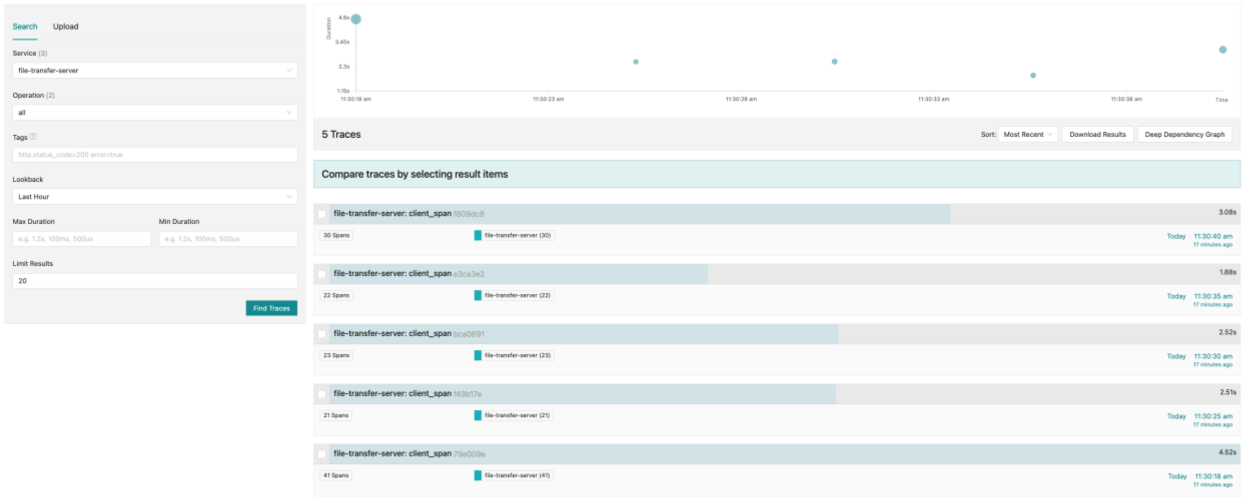
Tags

client_id	1763310635485
otel.scope.name	client.tracer
span.kind	internal
total_bytes	952963061
total_files	21
total_latency_seconds	1.8724987506866455

```
> Process: telemetry.sdk.language = python | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.37.0
v Logs (44)
> 1.63ms: event = Client 1763310635485 has connected to server
> 1.64ms: event = Sending file file_0.bin
> 175.73ms: event = Sent
> 175.73ms: event = Sending file file_1.bin
> 249.44ms: event = Sent
```

File-Transfer-Server

Server Trace Results



Search Upload

Service (3): file-transfer-server

Operation (2): all

Tags: http.status_code=200 error=true

Lookback: Last Hour

Max Duration: e.g. 1.2s, 100ms, 500us

Min Duration: e.g. 1.2s, 100ms, 500us

Limit Results: 20

Find Traces

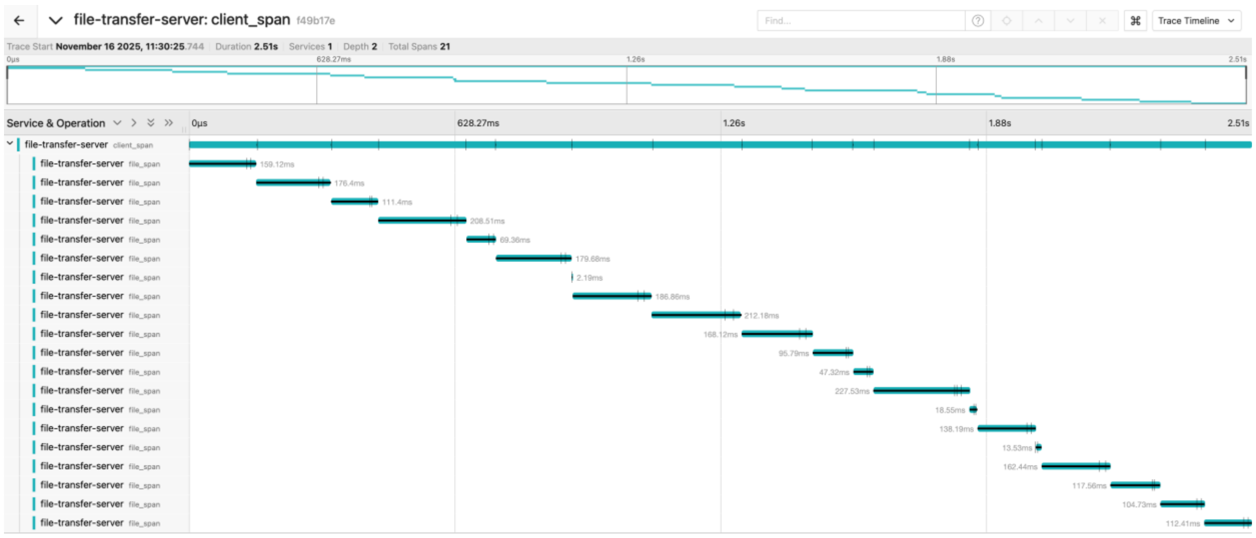
5 Traces

Sort: Most Recent | Download Results | Deep Dependency Graph

Compare traces by selecting result items

```
file-transfer-server: client_span f809dc8                                3.08s
30 Spans        file-transfer-server (30)             Today | 11:30:40 am
                                                           17 minutes ago
file-transfer-server: client_span e3ca3e2                                1.88s
22 Spans        file-transfer-server (22)             Today | 11:30:35 am
                                                           17 minutes ago
file-transfer-server: client_span bca0691                                2.52s
23 Spans        file-transfer-server (23)             Today | 11:30:30 am
                                                           17 minutes ago
file-transfer-server: client_span f49b17e                                2.51s
21 Spans        file-transfer-server (21)             Today | 11:30:25 am
                                                           17 minutes ago
file-transfer-server: client_span 79e009e                                4.52s
41 Spans        file-transfer-server (41)             Today | 11:30:18 am
                                                           17 minutes ago
```

File Transfer Server Trace View



file-transfer-server: client_span f49b17e

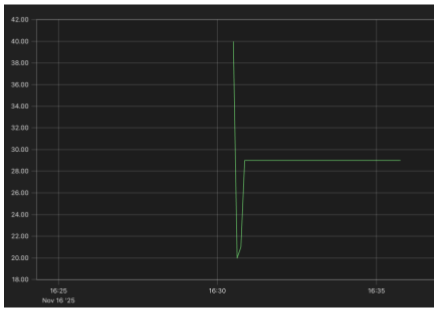
Trace Start **November 16 2025, 11:30:25**.744 | Duration **2.51s** | Services **1** | Depth **2** | Total Spans **21**

Trace Timeline

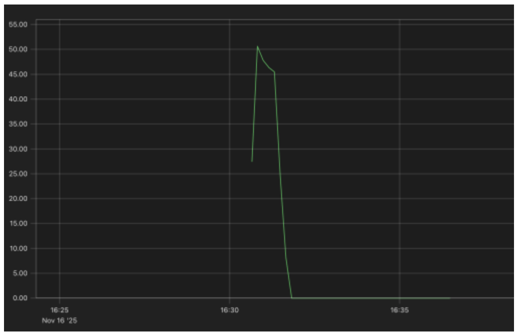
```
Service & Operation                     0μs    628.27ms    1.26s    1.88s    2.51s
file-transfer-server  client_span
  file-transfer-server  file_span   159.12ms
  file-transfer-server  file_span   176.4ms
  file-transfer-server  file_span   111.4ms
  file-transfer-server  file_span   208.51ms
  file-transfer-server  file_span   69.36ms
  file-transfer-server  file_span   179.68ms
  file-transfer-server  file_span   2.19ms
  file-transfer-server  file_span   186.86ms
  file-transfer-server  file_span   212.18ms
  file-transfer-server  file_span   168.12ms
  file-transfer-server  file_span   95.79ms
  file-transfer-server  file_span   47.32ms
  file-transfer-server  file_span   227.53ms
  file-transfer-server  file_span   18.55ms
  file-transfer-server  file_span   138.19ms
  file-transfer-server  file_span   13.53ms
  file-transfer-server  file_span   162.44ms
  file-transfer-server  file_span   117.56ms
  file-transfer-server  file_span   104.73ms
  file-transfer-server  file_span   112.41ms
```

Prometheus

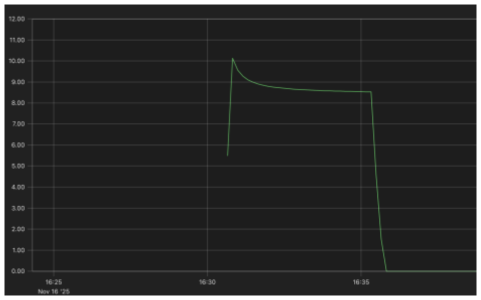
Files Generated Counter



Data Transfer (MB) Rate Over Time



Throughput (MB/S)



Results

Table 1: Client-Side Latency

Metric	AlwaysOn (100%)	25% Sampling	Difference
Average	2,889.60 ms (2.89s)	4,481.56 ms (4.48s)	+1,591.96 ms (+55.1%)
Minimum	1,872.47 ms (1.87s)	2,608.92 ms (2.61s)	+736.45 ms (+39.3%)
Maximum	4,505.43 ms (4.51s)	5,840.14 ms (5.84s)	+1,334.71 ms (+29.6%)
Median	2,520.09 ms (2.52s)	4,525.73 ms (4.53s)	+2,005.64 ms (+79.6%)
Std Dev	995.54 ms	1,217.80 ms	+222.26 ms (+22.3%)

Table 2: Server-Side Latency

Metric	AlwaysOn (100%)	25% Sampling	Difference
Average	2,903.81 ms (2.90s)	4,729.43 ms (4.73s)	+1,825.62 ms (+62.9%)
Minimum	1,877.40 ms (1.88s)	2,612.49 ms (2.61s)	+735.09 ms (+39.2%)
Maximum	4,521.10 ms (4.52s)	5,879.94 ms (5.88s)	+1,358.84 ms (+30.0%)
Median	2,524.79 ms (2.52s)	4,987.42 ms (4.99s)	+2,462.63 ms (+97.5%)
Std Dev	999.68 ms	1,178.10 ms	+178.42 ms (+17.8%)

Table 3: Per-File Operation Latency

Operation	AlwaysOn (100%)	25% Sampling	Difference
Client File Send (Avg)	109.35 ms	121.69 ms	+12.34 ms (+11.3%)
Client File Send (Median)	110.04 ms	127.52 ms	+17.48 ms (+15.9%)
Server File Receive (Avg)	109.93 ms	125.33 ms	+15.40 ms (+14.0%)
Server File Receive (Median)	111.91 ms	114.68 ms	+2.77 ms (+2.5%)

Table 4: File Throughput

Metric	AlwaysOn (100%)	25% Sampling	Difference
Average	9.27 files/sec	8.23 files/sec	-1.04 files/sec (-11.2%)
Minimum	8.01 files/sec	7.80 files/sec	-0.21 files/sec (-2.6%)
Maximum	11.22 files/sec	8.65 files/sec	-2.57 files/sec (-22.9%)

Table 5: Data Throughput

Metric	AlwaysOn (100%)	25% Sampling	Difference
Average (MB/sec)	461.99 MB/sec	423.71 MB/sec	-38.28 MB/sec (-8.3%)
Minimum (MB/sec)	443.87 MB/sec	412.79 MB/sec	-31.08 MB/sec (-7.0%)
Maximum (MB/sec)	485.36 MB/sec	448.34 MB/sec	-37.02 MB/sec (-7.6%)

Table 6: Network Throughput

Metric	AlwaysOn (100%)	25% Sampling	Difference
Average	3,695.95 Mbps	3,389.67 Mbps	-306.28 Mbps (-8.3%)
Minimum	3,550.98 Mbps	3,302.36 Mbps	-248.62 Mbps (-7.0%)
Maximum	3,882.85 Mbps	3,586.68 Mbps	-296.17 Mbps (-7.6%)

Table 7: Trace Collection & Visibility

Aspect	AlwaysOn (100%)	25% Sampling	Impact
Total Traces (Client)	10 traces	8 traces	-20% visibility
Total Traces (Server)	10 traces	8 traces	Matched client sampling
Total Spans (Client)	142 spans	192 spans	+35.2% more spans*
Avg Spans/Trace	14.2	24.0	+69.0%
Client Spans Captured	5/5 (100%)	5/20 (25%)	Expected sampling
Server Connections	5 handled	7 handled	More in 25% sample
File Send Spans	132	184	More files in sampled runs

Table 8: Standard Deviation Analysis

Metric	AlwaysOn	25% Sampling	Change
Client Latency StdDev	995.54 ms	1,217.80 ms	+22.3%
Server Latency StdDev	999.68 ms	1,178.10 ms	+17.8%

Table 9: Compression Performance

Aspect	AlwaysOn	25% Sampling
Files Compressed	132/132 (100%)	183/184 (99.5%)
Avg Compression Ratio	99.71%	99.71%
Min Compression Ratio	99.63%	99.71%
Max Compression Ratio	99.71%	99.71%

Table 1 (Client-Side Latency) demonstrates that 25% sampling shows higher latency results (with an average of +55.1%). Lower sampling should reduce overhead and improve performance. Table 2 (Server-Side Latency) shows a similar degradation pattern (+62.9% average), suggesting that both the client and server are affected. Table 3 presents the per-file operation latency, where individual file operations exhibit reasonable increases (ranging from 11 to 16%), which are significantly smaller than the overall latency differences.

Tables 4 through 6 show the throughput analysis of the files, data, and network. Throughput decreased by 8 to 11% with reduced sampling, which correlates with increased latency. Maximum throughput was reduced by 22.9%, indicating a decline in peak performance.

Table 7 analyzes the Total Traces and Spans across the server and client. In general, the 25% sampling method collected more total spans because the sampled runs transferred more files (184 vs 132). The 25% sampling captured only 25% of client executions (5 out of 20 runs), which happened to be larger workloads (36.8 vs. 26.4 files per run on average). When it comes to blind spots, 75% of executions have zero visibility, making it harder to detect issues.

With AlwaysOn Sampling, all errors and edge cases are captured on both the server and client, meaning complete visibility into every execution. However, there are higher storage requirements and double the telemetry data points.

With 25% sampling, storage is reduced by about 75% across both the client and server. Trace integrity is maintained, and there is lower export frequency throughout both services. However, it may miss rare errors or edge cases, and sample bias is a potential concern. If specific failed runs are not sampled, they cannot be debugged.

Table 8 compares the standard deviations of client latency and server latency. A 25% sampling shows much higher variance (22% for the client vs. 18% for the server). Performance is less predictable, likely due to workload differences rather than the sampling itself.

Table 9 shows that both configurations have excellent compression (a ratio of 99.71%). The high compression is due to the use of the Lorem Ipsum pattern for generating larger files in the test data.

Conclusion

In conclusion, workload is more significant than sampling (42.5% difference in data volume). Also, sampling appears to have reduced visibility, missing 75% of executions on both the client and server. This reduces debugging capability for distributed issues. Overall, overhead is minimal, with Table 3 suggesting that actual tracing overhead ranges from 5 to 10%. With the same sampling rate on both client and server, trace continuity is guaranteed and orphaned spans are prevented.

Part 2 Report

Introduction

We introduced a bug where 30% of large files (at least 10 MB) were not compressed at all (i.e. Z_NO_COMPRESSION instead of Z_BEST_COMPRESSION).

Statistical Debugging Predicates

The predicates we used were:

- bug_triggered: flag indicating bug was activated
- is_large_file: returns true if file size is greater than 10 MB
- compression_skipped: If ratio is around 0, then compression was not used
- file_size_huge: returns true if file size is greater than 50 MB
- file_size_medium: returns true if file size is between 1 and 10 MB
- file_size_small: returns true if file size is less than 1 MB
- encryption_overhead_high: returns true if encryption overhead is larger than 50 %
- is_compressed: returns true if compression was applied

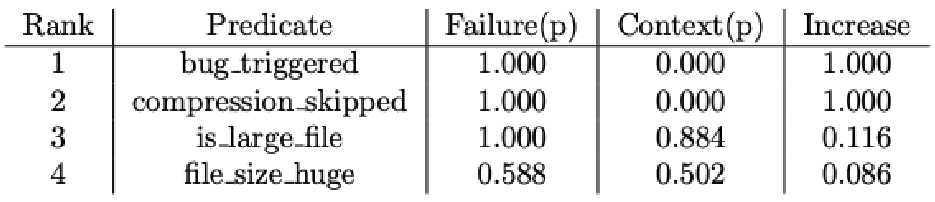
Our top predicates were:

1. bug_triggered: Present in all failures (100%) and absent in no successes (0%)
2. compression_skipped: Present in all failures (100%) and absent in no successes (0%)

These predicates are clearly the root cause of failures since they appear in all failures but no successes.

The predicate is_large_file appears in all failures (100%) but also in most successes (88.4%), confirming the bug only affects large files.

Results



Rank	Predicate	Failure(p)	Context(p)	Increase
1	bug_triggered	1.000	0.000	1.000
2	compression_skipped	1.000	0.000	1.000
3	is_large_file	1.000	0.884	0.116
4	file_size_huge	0.588	0.502	0.086

According to the table above, the predicates bug_triggered and compression_skipped both have an Increase metric of 1.000, indicating that these predicates triggered the failures. Furthermore, these predicates appear in 100% of failed operations (177/177) but in 0% of successful operations (0/500).

The predicate is_large_file appears in all failures but also in 88.4% of successes, confirming that while the bug only affects large files, not all large files fail: only those where the bug is triggered (30% probability).

Conclusion

The statistical debugging approach successfully identified our injected bug without prior knowledge of its location or nature, proving that this debugging technique is very efficient.