Maximizing Throughput in Distributed Systems: Java & Go Servers, Java Client

Random Team: Yinuo Feng, Nuanxin Jin, Boyuan Sun, Zihe Chen, Chenyu Jiang

Table of contents

Tech Stack

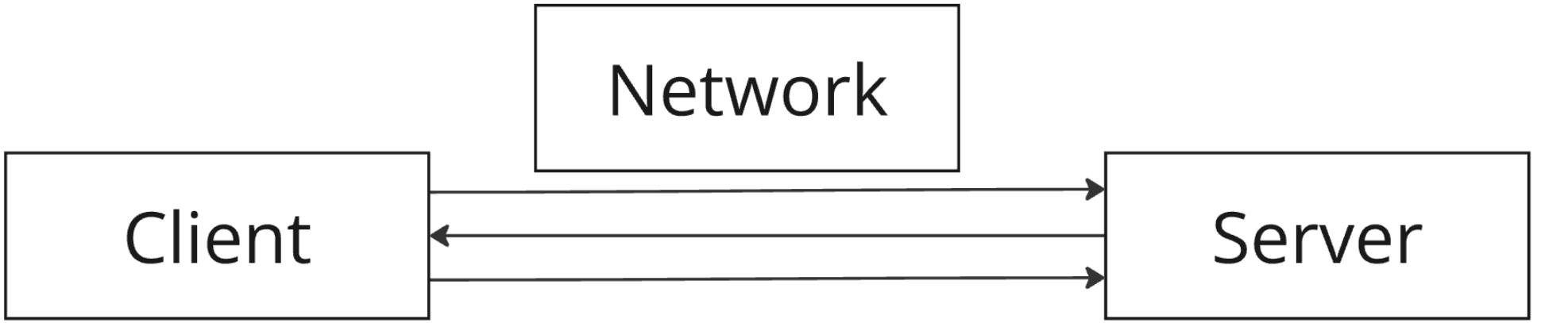
Servers: Java & Go
Http Client: Java

Infrastructure: AWS EC2, Tomcat

Testing & Metrics: Wall time, Throughput, Latency, p99 response time

How to maximize throughput?

Proposed System Architecture



- Bandwith

- Connection Pool
- Function Call
- Thread Create
- Thread Run
- Thread Close
- Delay

Runnable Method ★

NonRunnable Method with Executor and Scheduler

- Threads(MaxThread)
- Connection Pool
- Memory & GC
- CPU usage

Runnable Method

- Runnable Interface executed by Thread

```
private CloseableHttpClient client;  2 usages
private List<Row> data;  2 usages

public ClientGet(String IPAddr, CloseableHttpClient client, List<Row> d
  this.getUrl = "http://" + IPAddr + "/IGORTON/AlbumStore/1.0.0/albums/
  this.client = client;
  this.data = data;
}

// stolen from https://hc.apache.org/httpclient-legacy/tutorial.html
public void run() {

  // Create a method instance.
    CountDownLatch completed2 = new CountDownLatch(totalThreads);
    long start = System.currentTimeMillis();
    for (int j = 0; j < numGroups; j++) {
      for (int i = 0; i < groupSize; i++) {
        Runnable thread = () -> {
          for (int k = 0; k < 1000; k++) {
            clientGet.run();
            clientPost.run();
          }
          completed2.countDown();
          //System.out.println("Thread completed");
        };
        new Thread(thread).start();
        //System.out.println("Thread started");
      }
      if (j != numGroups - 1) {
        Thread.sleep( l: delay * 1000);
      }
    }
```

NonRunnable Method

- Simple Get and Post Function
- ThreadPoolExecutor
- Scheduler

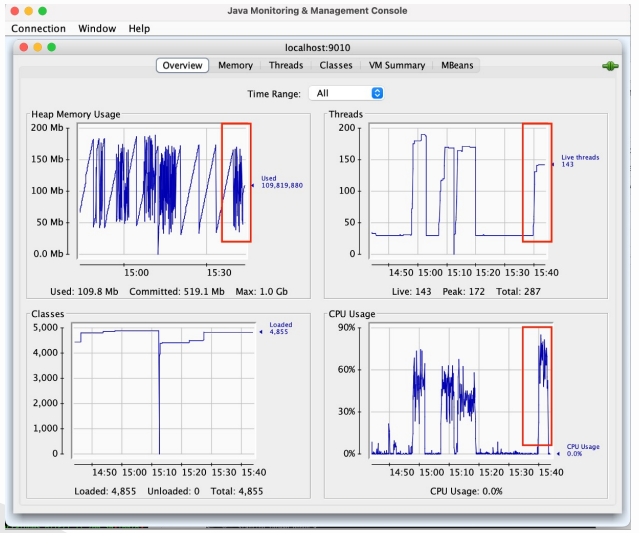
```
private static void sendGetRequest(int i) throws IOException {  2 usages

  long start = System.currentTimeMillis();
  HttpGet getMethod = new HttpGet( uri: IPAddrTomcat + "/" + i);
  try (CloseableHttpResponse response = client.execute(getMethod)) {
    EntityUtils.consume(response.getEntity());
  }
  long end = System.currentTimeMillis();
}
CountDownLatch latch2 = new CountDownLatch(threadGroupSize * numThreadGroups);
long start = System.currentTimeMillis();
for (int i = 0; i < numThreadGroups; i++) {
  int groupIndex = i;
  scheduler.schedule(() -> {
    for (int j = 0; j < threadGroupSize; j++) {
      int finalJ = j;
      executorService.submit(() -> {
        for (int k = 0; k < 1000; k++) {
          try {
            sendPostRequest();
            sendGetRequest( i: groupIndex * threadGroupSize + finalJ);
          } catch (IOException e) {
            throw new RuntimeException(e);
          }
        }
        latch2.countDown();
      });
    }
  }, delay: groupIndex * delay, TimeUnit.MILLISECONDS);
}
```

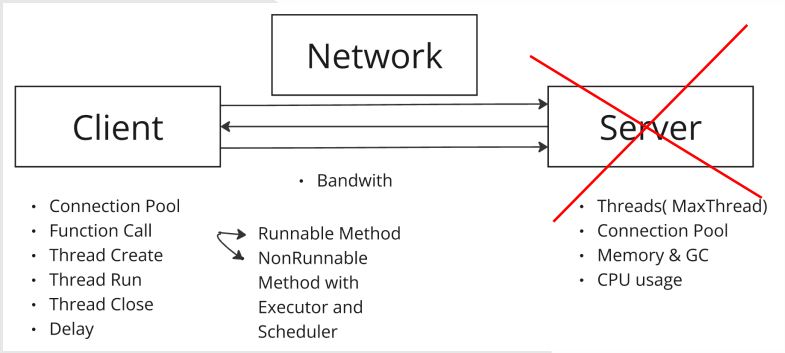
Experiments & Results

Biggest Problem – X Server

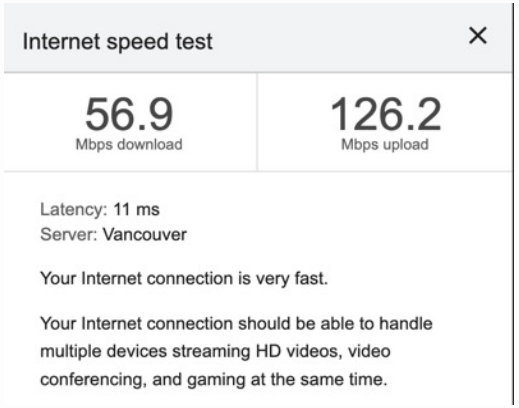
```
Time: 2025-02-12 09:44:12
Used:      136,516 kbytes
Committed: 506,944 kbytes
Max:     1,013,632 kbytes
GC time: 2.766 seconds on Copy (450 collections)
         0.081 seconds on MarkSweepCompact (1 collections)
```

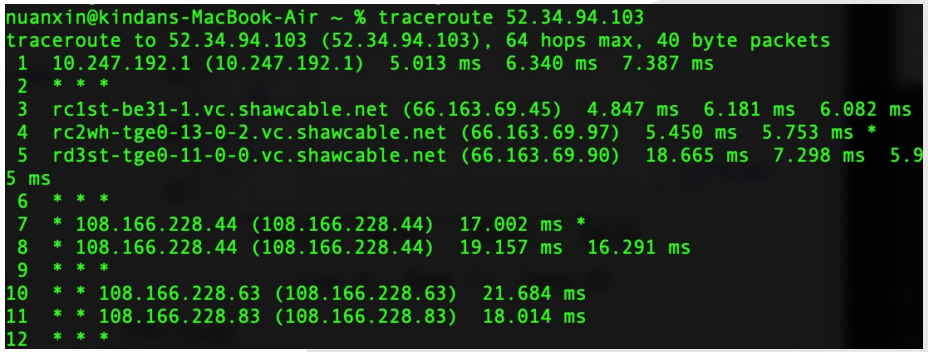


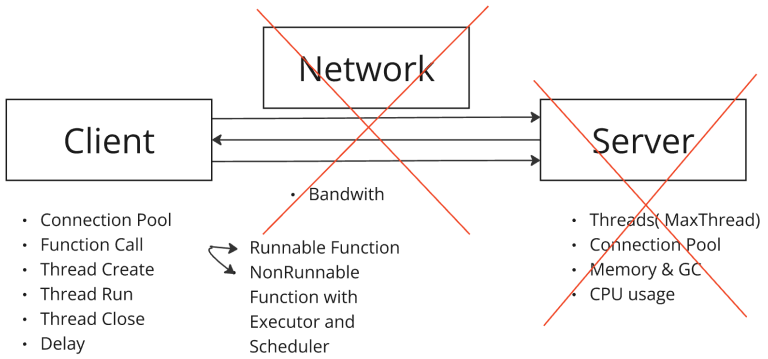

```
tcp6       0    207 172.31.28.40:8080       208.98.212.98:23432     ESTABLISHED
tcp6       0      0 172.31.28.40:8080       208.98.212.98:11907     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:51382     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:29608     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:5834      ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:27109     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:48708     ESTABLISHED
tcp6       0      0 172.31.28.40:8080       208.98.212.98:49893     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:27654     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:52310     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:16592     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:52143     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:28164     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:15015     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:29522     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:53808     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:21799     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:48719     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:49503     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:48478     ESTABLISHED
tcp6       0      0 172.31.28.40:8080       208.98.212.98:18808     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:50996     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:30638     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:5479      ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:58704     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:27059     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:52680     ESTABLISHED
tcp6       0      0 172.31.28.40:8080       208.98.212.98:13012     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:48540     ESTABLISHED
```



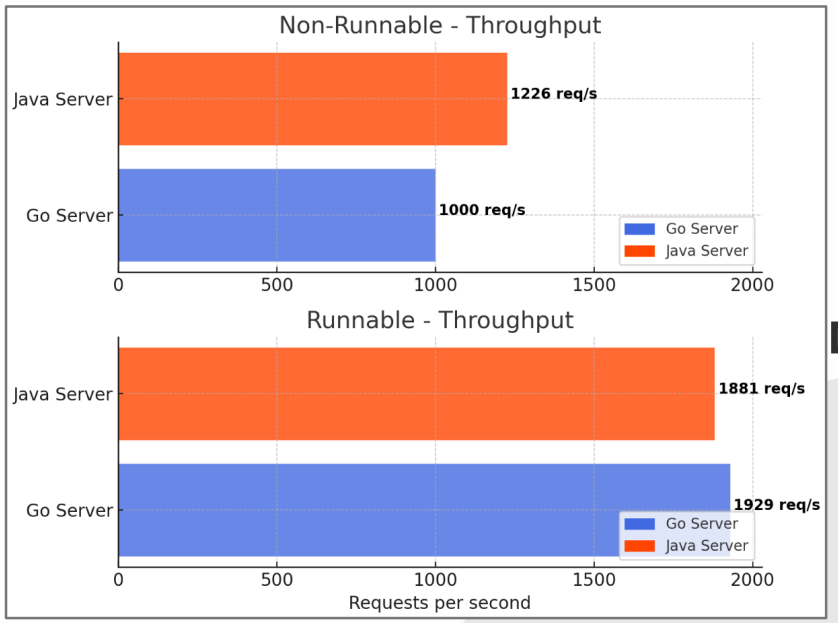
Biggest Problem – X Network

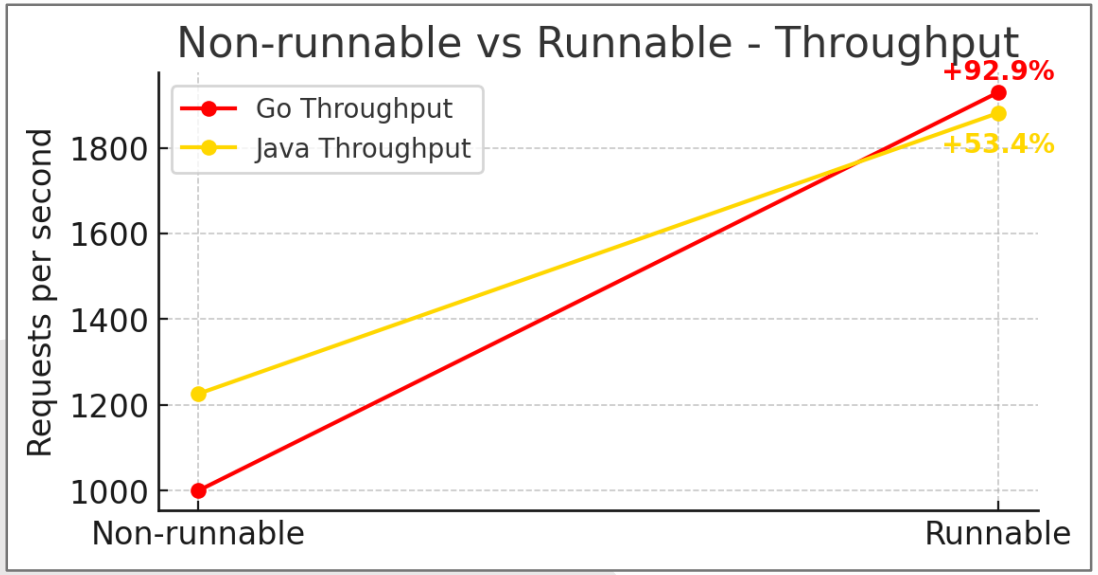






Runnable Increase Throughputs

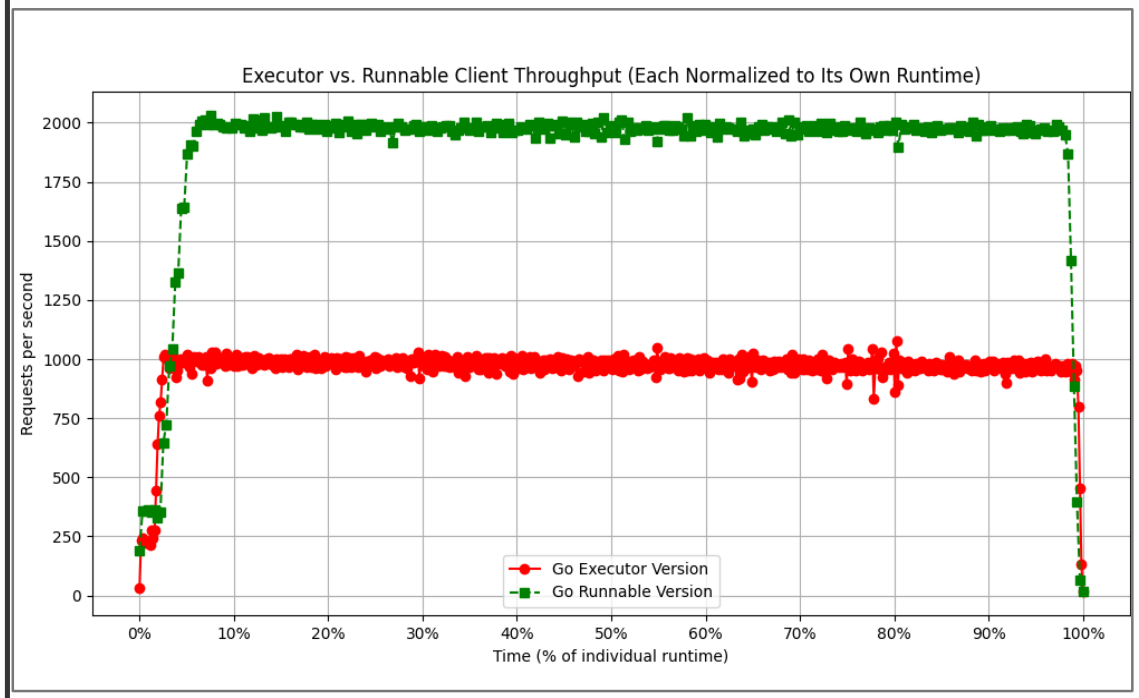




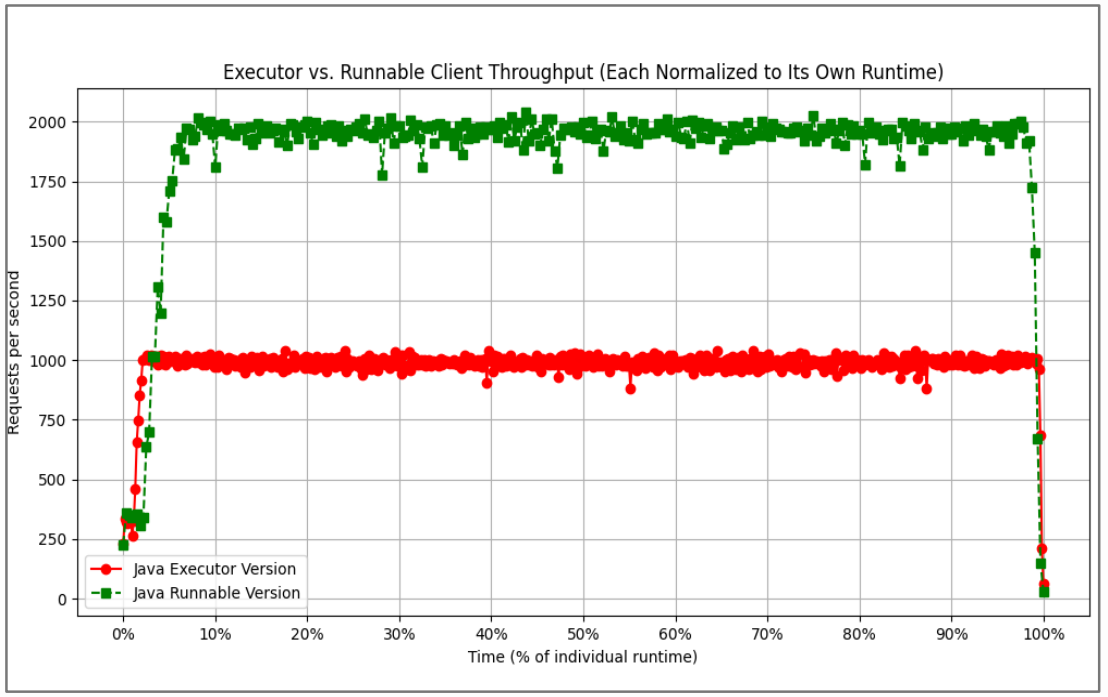
Throughput ↑ → Increased request handling capacity.

threadGroupSize = 10, numThreadGroups = 30, delay = 2

Runnable Increase Throughputs



Go Server

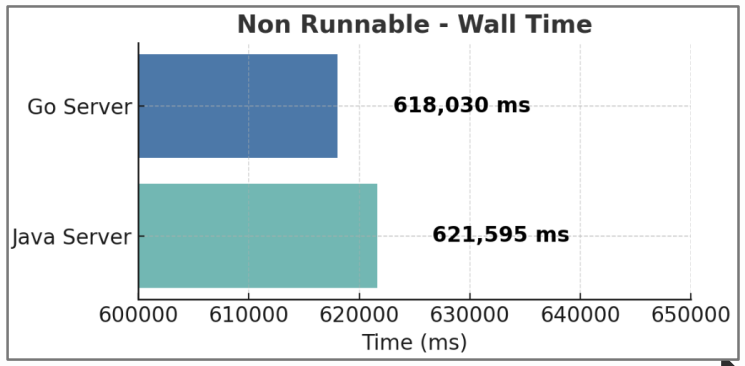


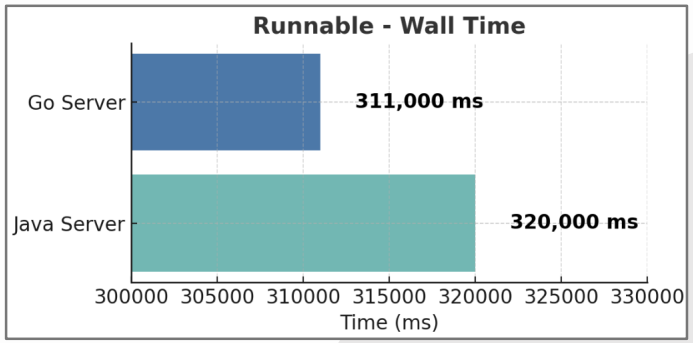
Java Server

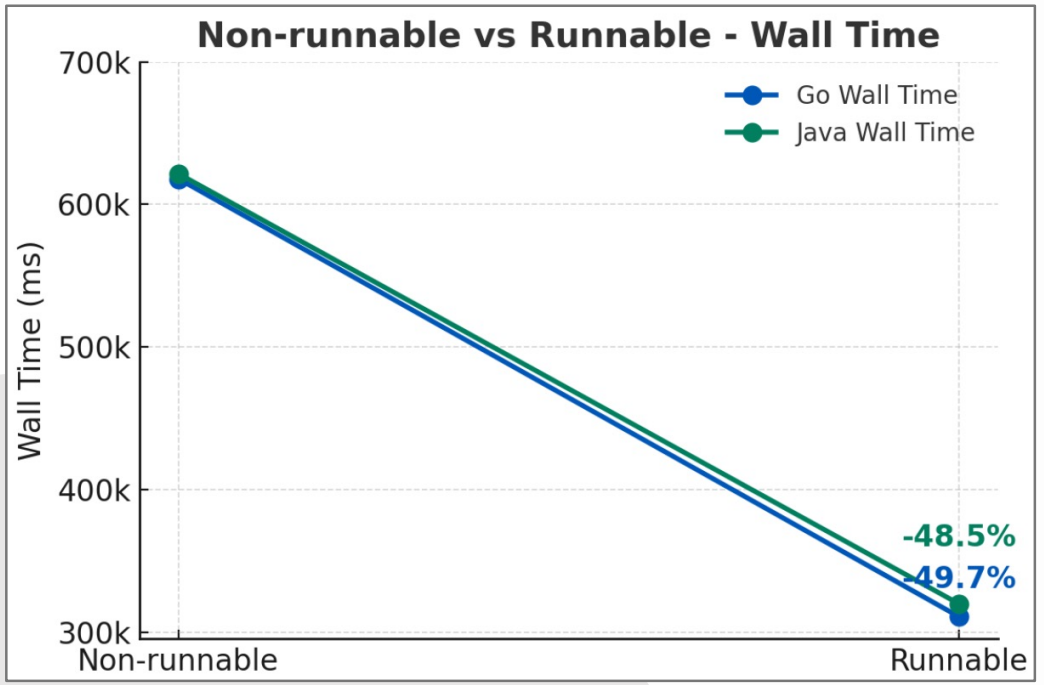
Runnable consistently achieves higher throughput

threadGroupSize = 10, numThreadGroups = 30, delay = 2

Runnable Reduce Wall Times



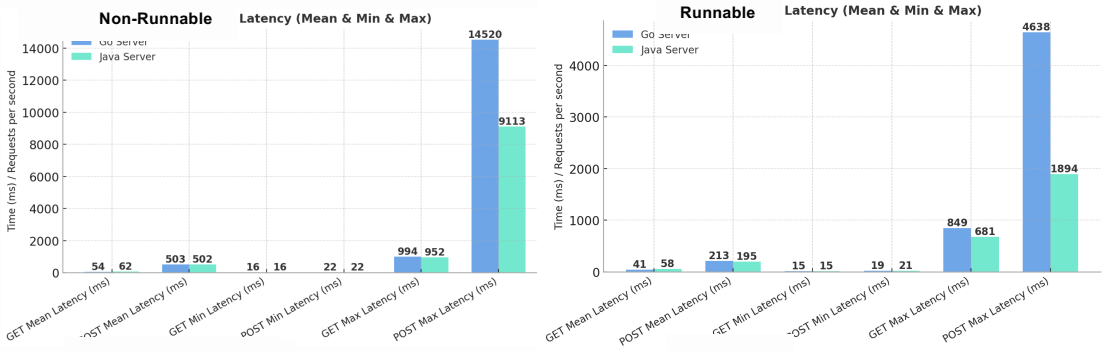


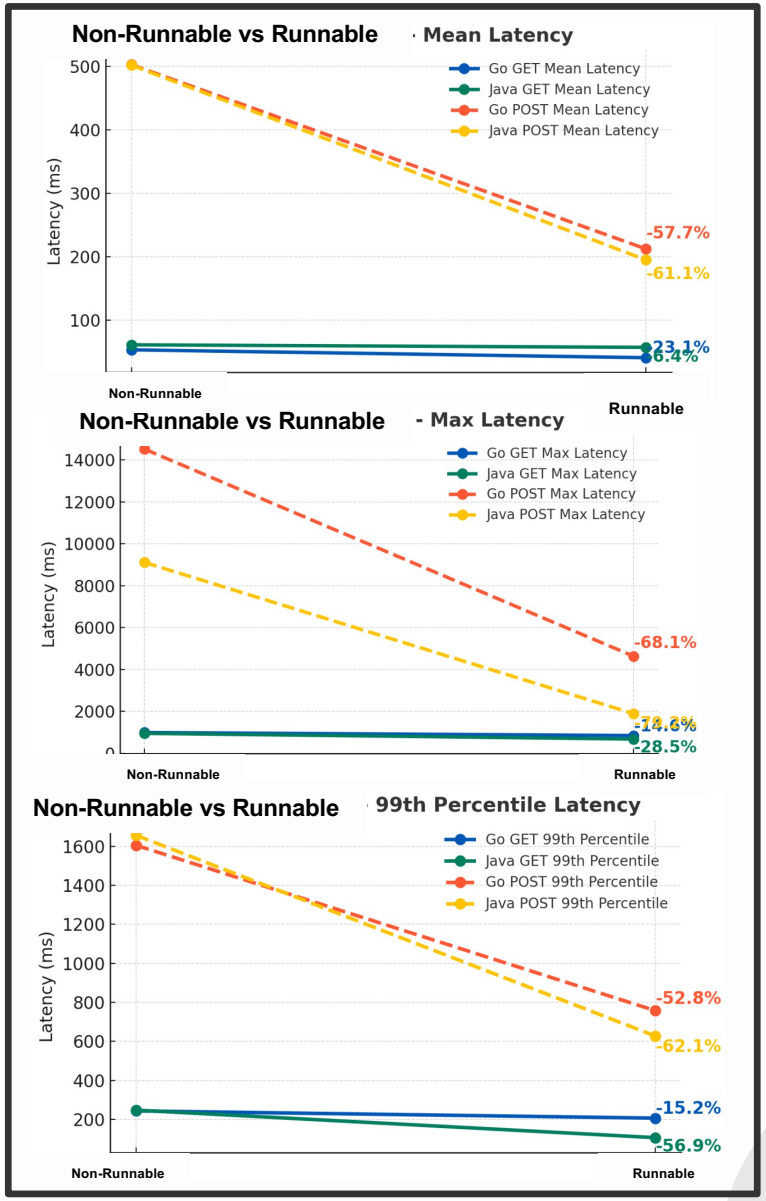


Wall Time ↓ → Faster execution and response times.

threadGroupSize = 10, numThreadGroups = 30, delay = 2

Runnable Reduce Latency





- **Mean Latency** ↓: Up to **61.1%**, with **POST requests improving the most**.
- **Max Latency** ↓: **68.1% (Go) and 79.2% (Java)** reduction, minimizing long-tail delays.
- **99th Percentile** ↓: Improved stability, with **up to 62.1%** fewer extreme slow requests.

threadGroupSize = 10, numThreadGroups = 30, delay = 2

Why Runnable

Client

- Connection Pool
- Function Call
- Thread Create
- Thread Run
- Thread Close
- Delay

```
Wall Time: 163560ms
Throughput: 1226 requests/second
```

```
Wall Time: 168463ms
Throughput: 1190 requests/second
```

getActiveCount()
getQueue().size()
getCompletedTaskCount()

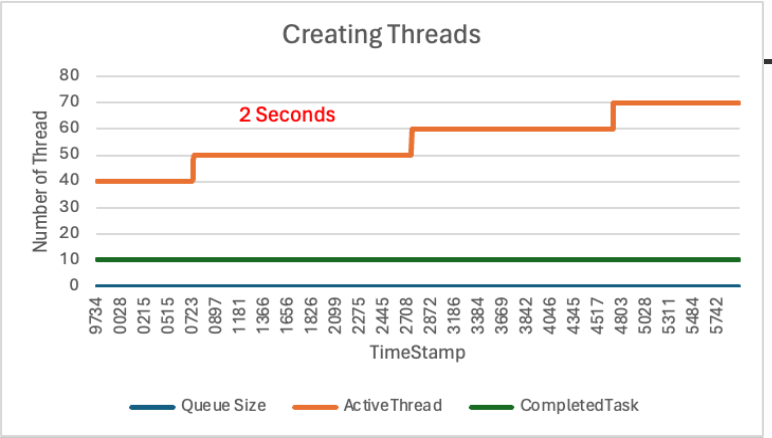
```
tcp6       0    201 172.31.28.40:8080       208.98.212.98:14638     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:28850     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:19719     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:36708     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:35073     ESTABLISHED
tcp6       0      0 172.31.28.40:8080       208.98.212.98:45511     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:41689     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:34710     ESTABLISHED
tcp6       0    207 172.31.28.40:8080       208.98.212.98:43824     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:26099     ESTABLISHED
tcp6       0      0 172.31.28.40:8080       208.98.212.98:17760     ESTABLISHED
tcp6       0    201 172.31.28.40:8080       208.98.212.98:49679     ESTABLISHED
```

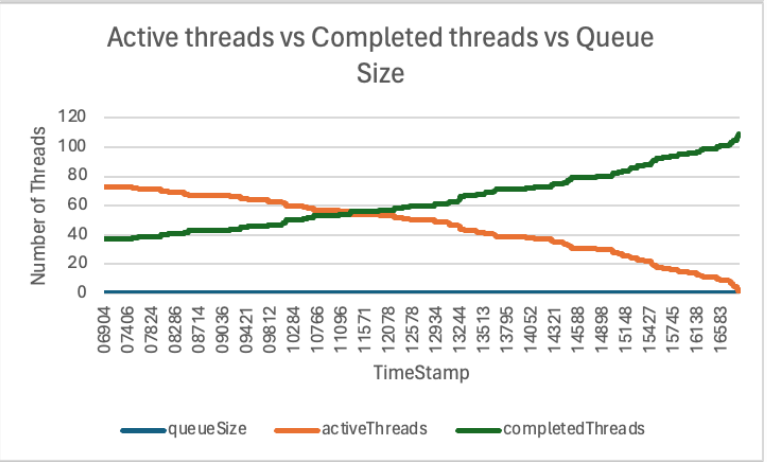
Runnable Method

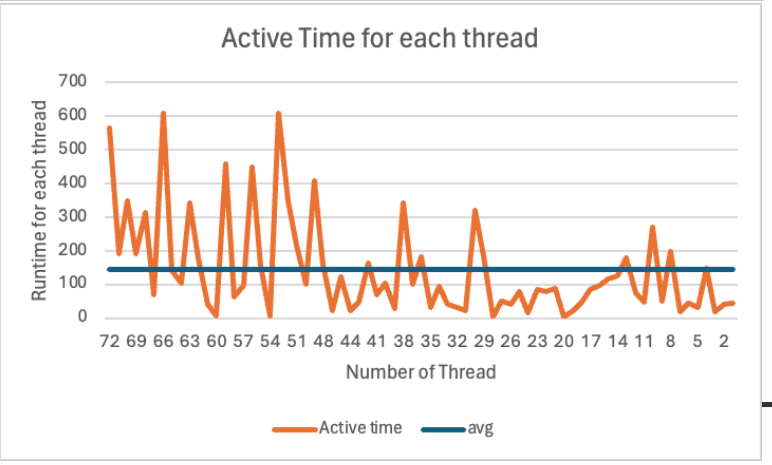
- NO Pool
- Runnable threads
- Thread Create instantly
- Thread Run instantly
- Thread Close automatically

NonRunnable Method

- Fixed Pool > requirement Or CachedPool
- Simple Function
- Thread Create
  Managed by executor + Scheduler
- Thread Run
  Managed by executor + Scheduler
- Thread Close
  Manually

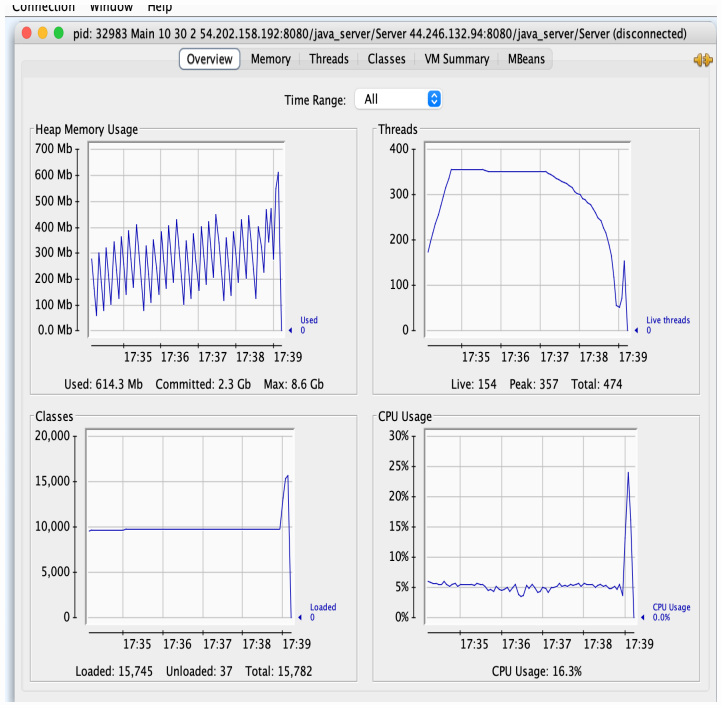




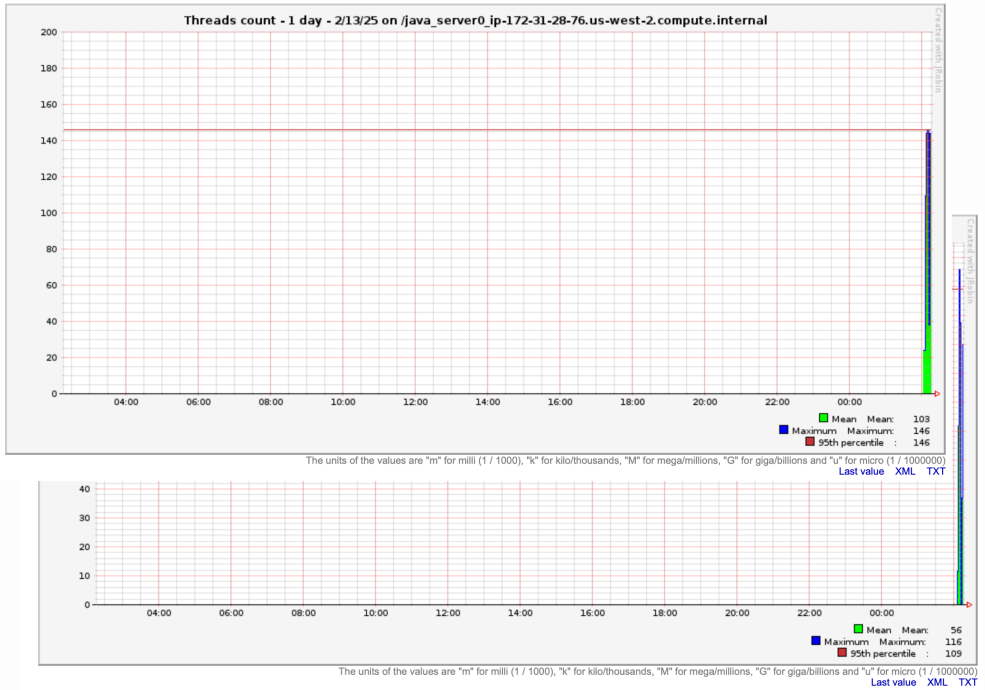


Chasing on Server

Scaling up server - How about launching another EC2?



Client: 300 threads



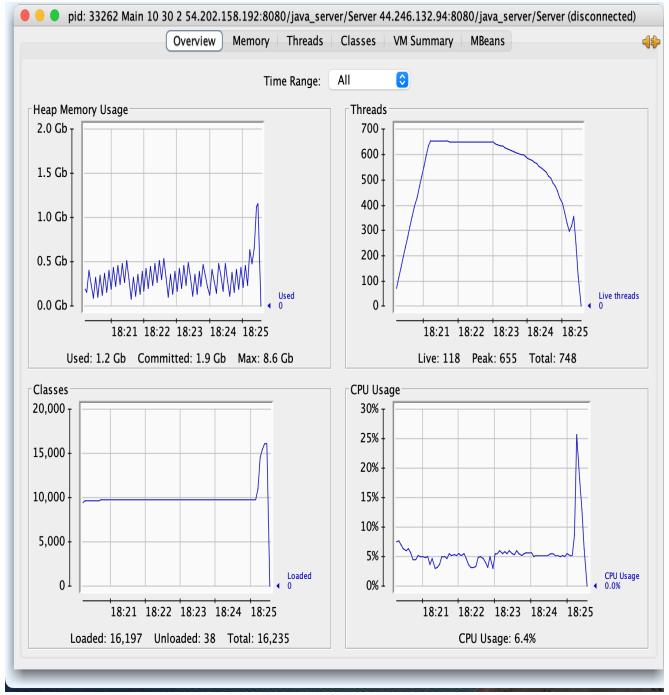
100+ threads / EC2

```
Time taken: 311s
Throughput: 1929 requests/s
SLF4J: Failed to load class "org.slf4j.
SLF4J: Defaulting to no-operation MDCAd
SLF4J: See http://www.slf4j.org/codes.h
GET Mean Latency: 39.72816611295681
POST Mean Latency: 213.74947508305647
GET Min Latency: 15
POST Min Latency: 21
GET Max Latency: 508
POST Max Latency: 3011
GET 50th Percentile: 37.0
POST 50th Percentile: 177.0
GET 99th Percentile: 201.0
POST 99th Percentile: 804.0
```

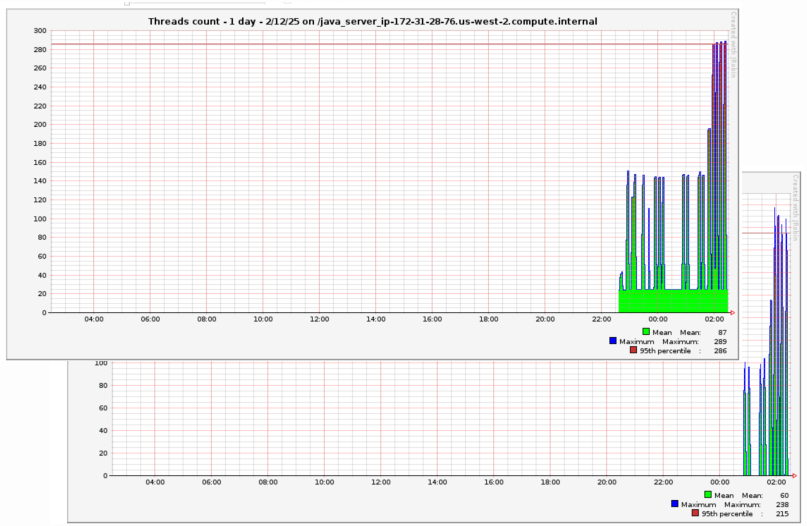
Throughput ~5%↑

Chasing on Server

More Threads from the Client!



Client: 600 threads



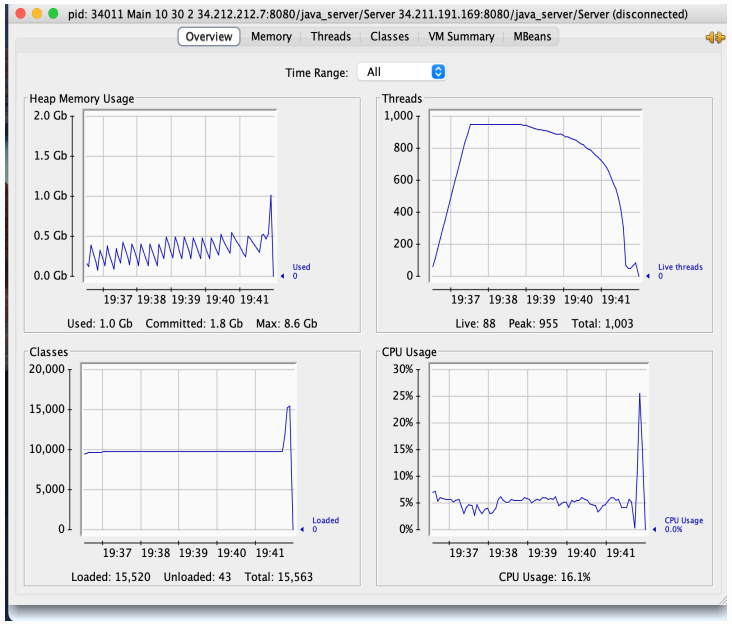
250+ Threads / EC2

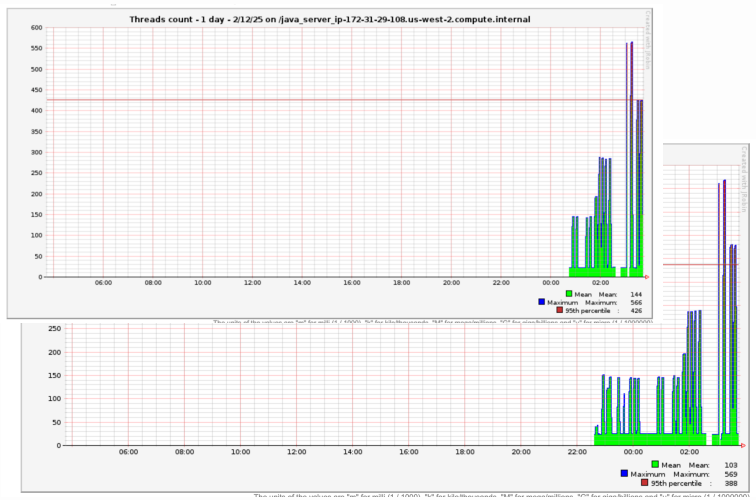
```
Time taken: 295s
Throughput: 2033 requests/s
SLF4J: Failed to load class "org.slf4j.
SLF4J: Defaulting to no-operation MDCAd
SLF4J: See http://www.slf4j.org/codes.h
GET Mean Latency: 45.847362433754306
POST Mean Latency: 463.3617679738785
GET Min Latency: 15
POST Min Latency: 21
GET Max Latency: 1143
POST Max Latency: 15355
GET 50th Percentile: 38.0
POST 50th Percentile: 370.0
GET 99th Percentile: 214.0
POST 99th Percentile: 1663.0
```

Throughput ~5% ↑

Chasing on Server

- Even More Threads!
- Throughput not improved
- POST Mean Latency Increased






```
Time taken: 308s
Throughput: 1946 requests/s
SLF4J: Failed to load class "org.slf4j.i
SLF4J: Defaulting to no-operation MDCAda
SLF4J: See http://www.slf4j.org/codes.ht
GET Mean Latency: 46.7095942800133
POST Mean Latency: 698.6709078816095
GET Min Latency: 15
POST Min Latency: 21
GET Max Latency: 1651
POST Max Latency: 43764
GET 50th Percentile: 39.0
POST 50th Percentile: 551.0
GET 99th Percentile: 223.0
POST 99th Percentile: 2543.0
```

Client 900 Threads

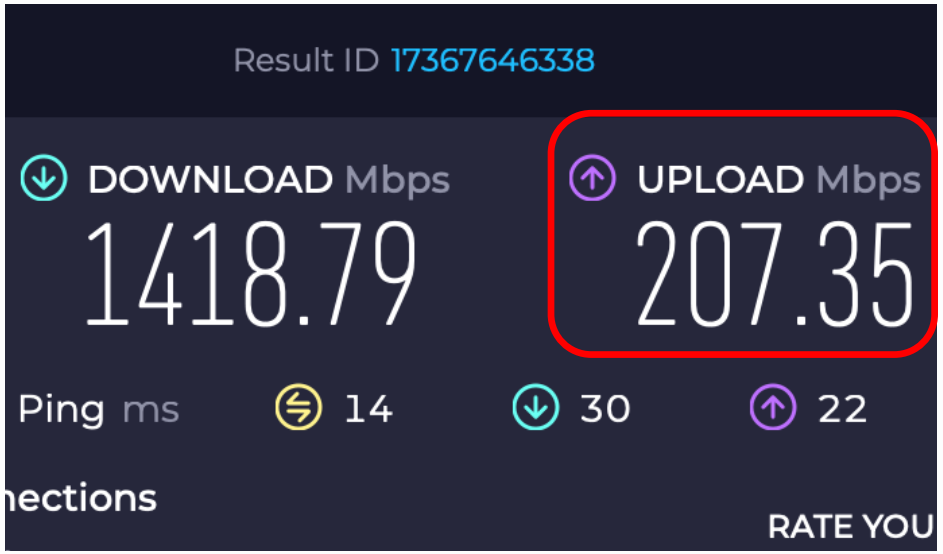
~450 Threads/ EC2

Throughput ☹️

Chasing on Network

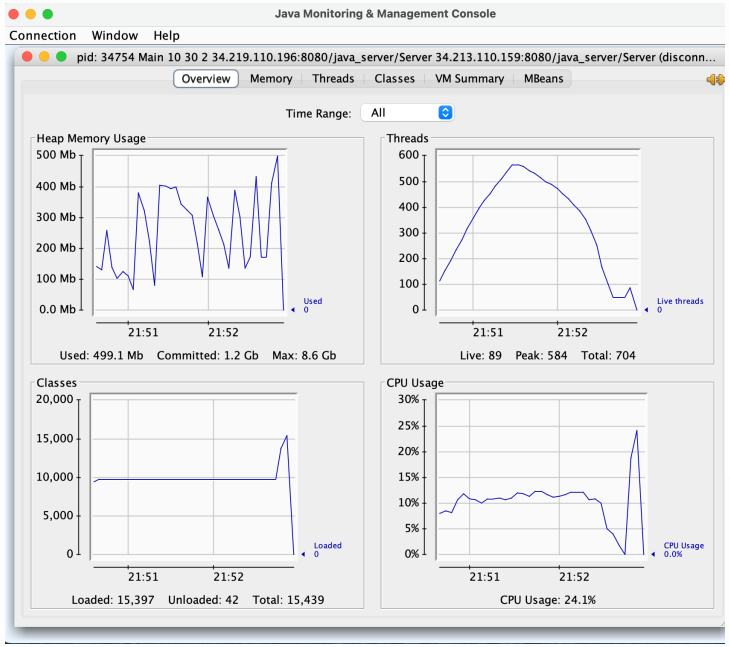
25KB Each

25KB x 1,000/s = 25MB/s

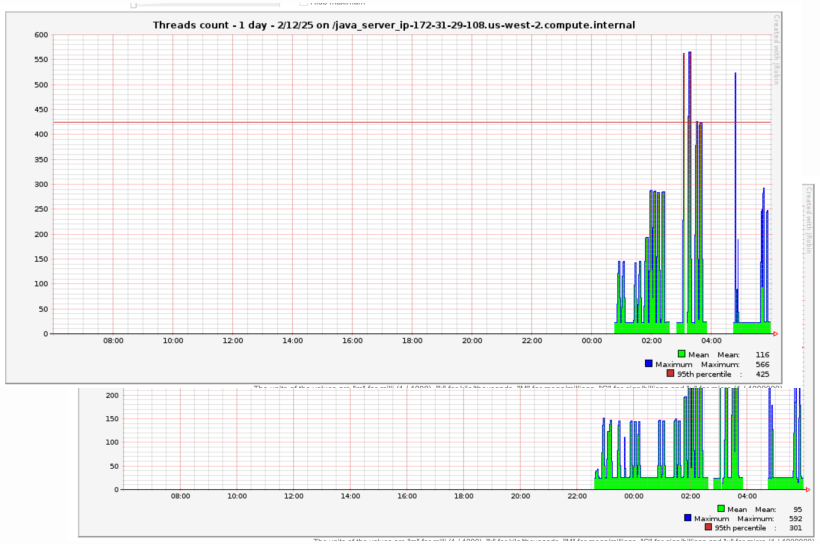


Chasing on Network

- Smaller Pic (2KB)!
- Throughput is more than doubled!



Client 600 Threads



EC2 ~250 Threads each

```
Time taken: 129s
Throughput: 4651 requests/s
SLF4J: Failed to load class "org.slf4j.im
SLF4J: Defaulting to no-operation MDCAdap
SLF4J: See http://www.slf4j.org/codes.htm
GET Mean Latency: 43.72685382059801
POST Mean Latency: 90.64286046511629
GET Min Latency: 16
POST Min Latency: 20
GET Max Latency: 432
POST Max Latency: 859
GET 50th Percentile: 37.0
POST 50th Percentile: 75.0
GET 99th Percentile: 112.0
POST 99th Percentile: 316.0
```

Throughput Doubled!

Chasing the Bottleneck

- Next Bottleneck: File Descriptors! Too many Files Open!

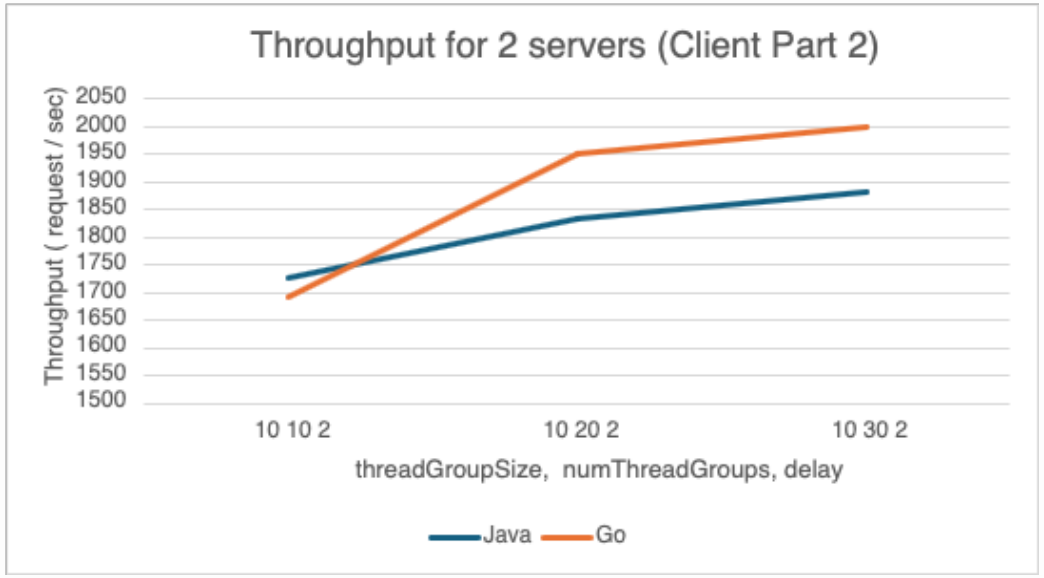
```
2/12/25, 4:53:56 AM SEVERE: Servlet.service() for servlet [Server] in context with path [/java_server] threw exception
2/12/25, 4:53:56 AM SEVERE: Servlet.service() for servlet [Server] in context with path [/java_server] threw exception
2/12/25, 4:53:56 AM SEV org.apache.tomcat.util.http.fileupload.impl.IOFileUploadException: Processing of multipart/form-data request failed. /usr/share/apache-tomcat-
2/12/25, 4:53:56 AM SEV 9.0.80/work/Catalina/localhost/java_server/upload_76d43006_6430_4ee6_80f2_e4c3d37d2750_00872851.tmp (Too many open files)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.http.fileupload.FileUploadBase.parseRequest(FileUploadBase.java:321)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.connector.Request.parseParts(Request.java:2838)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.connector.Request.getParts(Request.java:2739)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.connector.RequestFacade.getParts(RequestFacade.java:796)
2/12/25, 4:53:56 AM SEV     at javax.servlet.http.HttpServletRequestWrapper.getParts(HttpServletRequestWrapper.java:326)
2/12/25, 4:53:56 AM SEV     at Server.doPost(Server.java:57)
2/12/25, 4:53:56 AM SEV     at javax.servlet.http.HttpServlet.service(HttpServlet.java:555)
2/12/25, 4:53:56 AM SEV     at javax.servlet.http.HttpServlet.service(HttpServlet.java:623)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:209)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:153)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:178)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:153)
2/12/25, 4:53:56 AM SEV     at net.bull.javamelody.MonitoringFilter.doFilter(MonitoringFilter.java:239)
2/12/25, 4:53:56 AM SEV     at net.bull.javamelody.MonitoringFilter.doFilter(MonitoringFilter.java:215)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:178)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:153)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:168)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:90)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:481)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:130)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:93)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.java:670)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:74)
2/12/25, 4:53:56 AM SEV     at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:343)
2/12/25, 4:53:56 AM SEV     at org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:390)
2/12/25, 4:53:56 AM SEV     at org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:63)
2/12/25, 4:53:56 AM SEV     at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:926)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1790)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:52)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.threads.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1191)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.threads.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:659)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
2/12/25, 4:53:56 AM SEV     at java.base/java.lang.Thread.run(Thread.java:829)
2/12/25, 4:53:56 AM SEV Caused by: java.io.FileNotFoundException: /usr/share/apache-tomcat-9.0.80/work/Catalina/localhost/java_server/upload_76d43006_6430_4ee6_80f2_e4c3d37d2750_00872851.tmp (Too many open files)
2/12/25, 4:53:56 AM SEV     at java.base/java.io.FileOutputStream.open0(Native Method)
2/12/25, 4:53:56 AM SEV     at java.base/java.io.FileOutputStream.open(FileOutputStream.java:298)
2/12/25, 4:53:56 AM SEV     at java.base/java.io.FileOutputStream.<init>(FileOutputStream.java:237)
2/12/25, 4:53:56 AM SEV     at java.base/java.io.FileOutputStream.<init>(FileOutputStream.java:187)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.http.fileupload.DeferredFileOutputStream.thresholdReached(DeferredFileOutputStream.java:151)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.http.fileupload.ThresholdingOutputStream.checkThreshold(ThresholdingOutputStream.java:200)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.http.fileupload.ThresholdingOutputStream.write(ThresholdingOutputStream.java:126)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.http.fileupload.util.Streams.copy(Streams.java:104)
2/12/25, 4:53:56 AM SEV     at org.apache.tomcat.util.http.fileupload.FileUploadBase.parseRequest(FileUploadBase.java:317)
2/12/25, 4:53:56 AM SEV     ... 33 more
2/12/25, 4:53:56 AM SEVERE: Servlet.service() for servlet [Server] in context with path [/java_server] threw exception
```

Future Works

Future Experiments with Clients

- Executor introduces overhead?
- Eager Thread Creation vs. Lazy Thread Creation
- Older threads delaying newer threads?
- ……

Future Experiments with Server



Go Outperforms Java

Gorountine	Java thread
Managed by go runtime	Managed by operating system, kernel
Can be resized (like dynamic array in C!)	Fixed size
Initial stack size ~2KB	~1MB by default
Low overhead :)	High overhead :(

Future Experiments with Server - Scaling

Kubernetes

automates the deployment, scaling, and management of containerized applications

Auto Scaling Group

automatically adjust the number of instances based on traffic demand

Thanks!