Scala vs Clojure

performance under load

This blog is all about comparing performance metrics from the Clojure version against the Scala version of the functionally identical news feed service. See my previous blog on how these two micro-services compare in terms of code.

In both micro-services, I used the same library dependencies as much as possible, including version. I used the same EC2 instance set up and configuration. This means 5 m3.large instances (one each for Cassandra, Redis, Kafka, Solr, and the news feed micro-service itself) and a db.m3.large for the SQL DB with 100GB storage.

I used the same load test scenarios. This means the same load test application running 10 threads each on 5 t2.medium instances started 5 minutes apart. The throughput and latency metrics were of the outbound post activity.

When I blogged about the Scala news feed service last, I based the service on an open source project called Spray. I liked Spray because its routing based approach is more appropriate for micro-service development. Spray uses Akka Actors for its concurrency control. Typesafe is the company behind both Scala and Akka so there is a lot of love between the two. It seemed like a smart choice at the time.

Also, I was caching prepared statements in the Scala service. Both the Clojure and the Scala implementations depend in the open source c3p0 connection pooling library. It turns out that the c3p0 approach to connection pooling is not conducive to caching prepared statements so I stopped doing that.

After looking at the poor performance metrics and after profiling the application, I could see right away that I was going to have to move off of Spray.

It turns out that there is a serious bug in the Spray framework that prevents high throughput. Like I said earlier, Spray depends on Akka for its concurrency. You can configure Akka to use either Java thread executor pools or fork join pools as its dispatch mechanism. In my humble experience, executor pools are more efficient than fork join pools when it comes to I/O bound processing. Fork join pools are the default dispatcher for Akka. Although this project was configured to use executor pools, it was very clear when profiling the service that it was using the fork join pools anyway.

There is a serious bug in the Spray framework.

I wanted to replace Spray with a web framework that was battle tested in a high volume production environment. Twitter developed the Finatra open source project that runs on top of their Finagle project that integrates with Netty. There have been a lot of blogs recently that advocate for Netty over Jetty because of its use of asynchronous I/O. Twitter claims that they use Finatra a lot in the 100s of services that make up the Twitter production environment.

After making those code changes and another round of load tests, I found that Finatra throughput was about 3 times better than Spray. With both Finatra and Spray, you see better throughput at the beginning but it slides down to a quarter of the original performance and stays there after that. For Finatra, that is about 33 requests per second.

Except at the very beginning (which was expected), Finatra latency was very consistent with a mean of 11 ms, a median of 10 ms, and the 95th percentile at 22 ms per request.

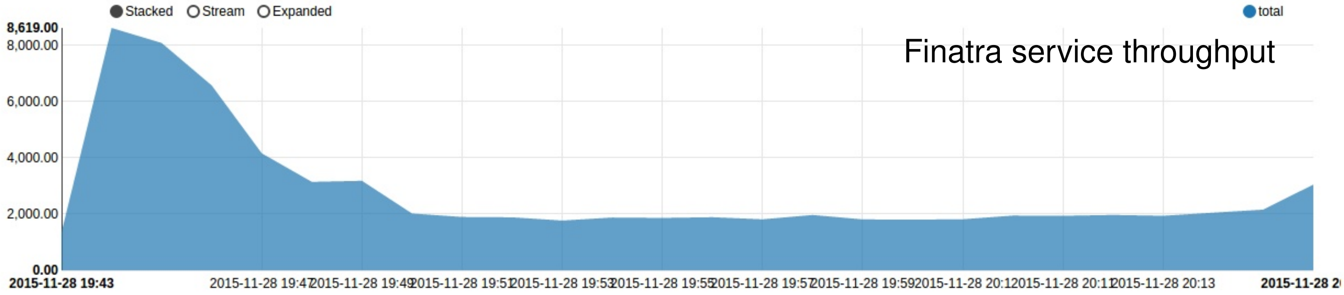
Profiling the Scala micro-service gave some good clues as to what was happening. CPU utilization was 75%. GC activity was 8.1%. Max heap memory was the highest at almost 2GB. There were 62 threads actively operating at peak load.

It turns out that Finatra and Netty aren't all that efficient because 40% of the time was spent there instead of the application specific code. Because of these inefficiencies, we saw only 129 write IOPS to the MySql DB.

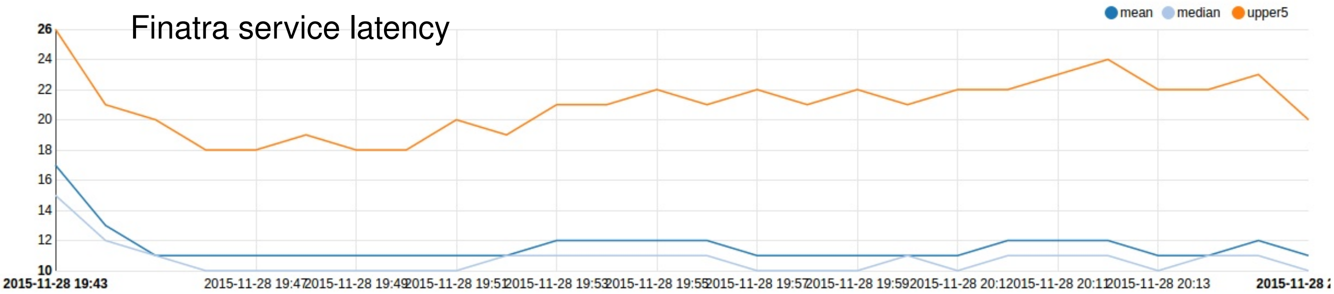
Here are the web service dependencies for the Clojure news feed service. I used Compojure for the routing rules. Compojure sits on top of Ring which can be its own stand-alone web container or be configured to integrate with Jetty. I used the Jetty integration.

Throughput for the Clojure service was very consistent. Within 5 minutes, it had climbed to a very steady 100 requests per second.

http://glennengstrand.info/software/architecture/oss/clojure



Finatra service throughput



Latency, however, was not consistent. It started pretty small but would jump up every 10 minutes and was spiky. The mean ended up at around 200 ms. The median was 186 ms. The 95th percentile was 315 ms per request.

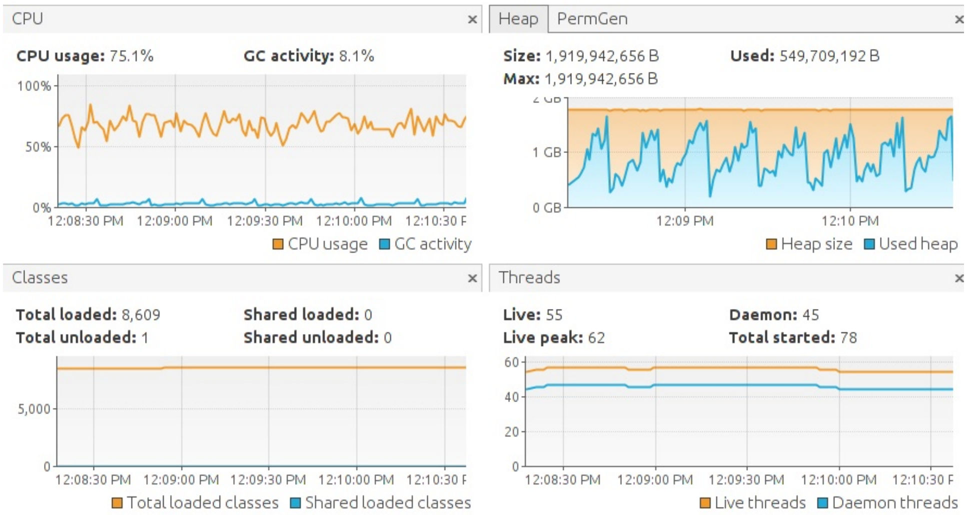
Profiling the Clojure micro-service revealed why its performance was so much better than the Scala service. It had the highest CPU utilization at 97% and the lowest GC activity at 6.9%. It used 30% more threads and 8 times less heap memory than the Scala service. Only 21% of the total time was spent in the web framework, half that of the Scala version. This resulted in 346 write IOPS to the MySql DB, almost 3 times more than the Scala service.

Although latency in the Clojure service was higher than in the Scala service, this is normal. It is expected that a service capable of running at a higher throughput would have more latency.

The write IOPS metric on the MySql DB was the most accurate predictor of throughput.

Why was the Scala version not capable of reaching the higher CPU utilization found in the Clojure version? Typically, a low CPU % is indicative of excessive synchronization.

Finatra and Netty aren't all that efficient because 40% of the time was spent there instead of the application code.



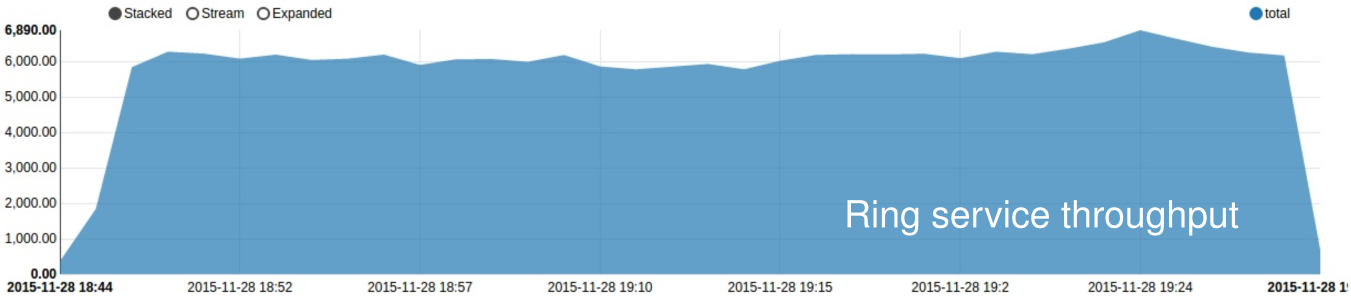
CPU usage: 75.1% **GC activity:** 8.1%

Size: 1,919,942,656 B **Used:** 549,709,192 B
Max: 1,919,942,656 B

Total loaded: 8,609 **Shared loaded:** 0
Total unloaded: 1 **Shared unloaded:** 0

Live: 55 **Daemon:** 45
Live peak: 62 **Total started:** 78

Finatra service profile



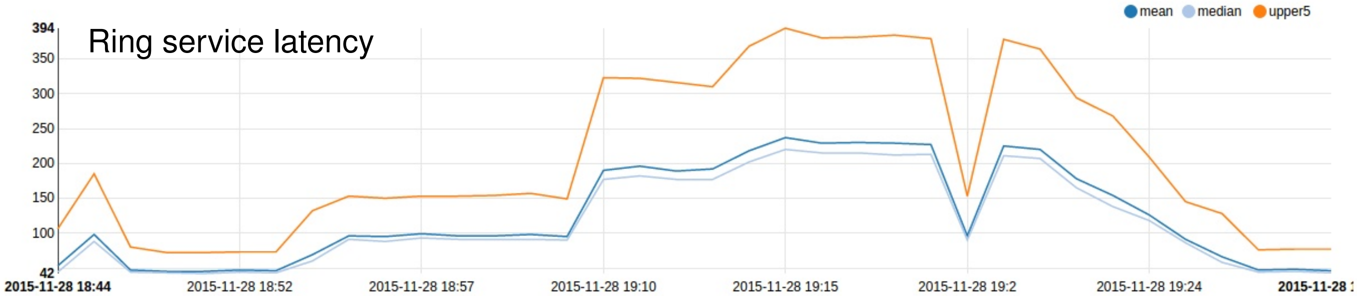
I introduced this blog as comparing two programming languages, Scala and Clojure, through load test performance. It really is about comparing two technology stacks. What I discovered in this learning adventure was actually quite interesting.

Consensus wisdom dictates that Netty is better than Jetty when it comes to throughput because you always have to have a thread dedicated to each request in Jetty whereas you don't in Netty. In Jetty, the throughput will always be constrained to the max number of threads that the server can have without excessive task switching which depends on the number of CPU cores that the server has.

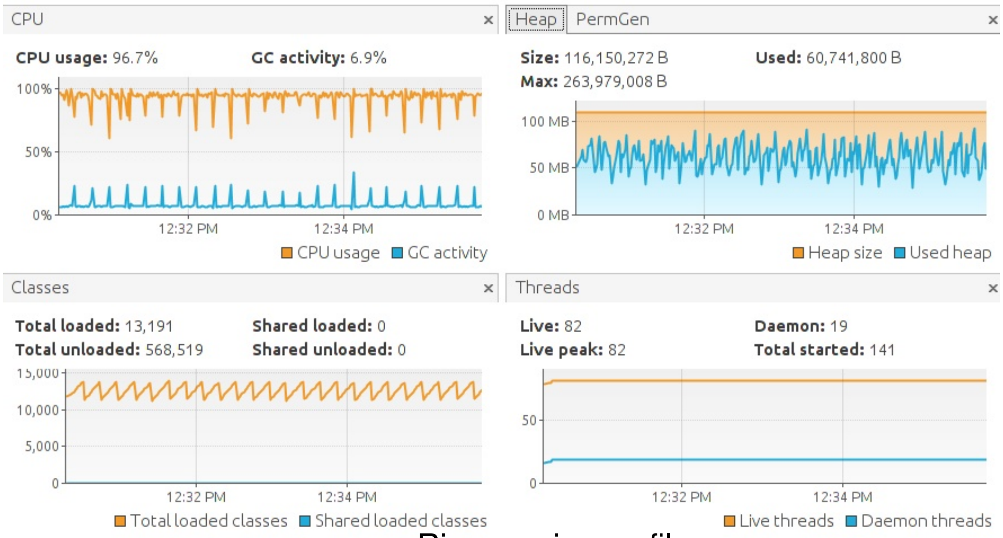
In order to be functionally equivalent to the Clojure version, you always had to return the results in the response to the request. In Netty, a long running task can actually kill the whole service precisely because it is not one thread per request. Because of that, I had to run all I/O bound processing in a separate thread pool. This means more overhead and higher synchronization.

Finatra is actively promoted by Twitter both online and in technology conferences. I learned about Ring and Compojure through an O'Reilly book on Clojure. There are basically two primary commiters to Compojure and Ring and not much promotion. In the end, Ring performed better than Finatra. I guess the lesson learned here is that the open source project with the biggest marketing budget isn't always the best technology.

http://glennengstrand.info/software/architecture/oss/scala



Clojure had highest CPU, lowest GC, more threads, and less memory resulting in 3 times more capacity than Scala.



CPU usage: 96.7% **GC activity:** 6.9%

Size: 116,150,272 B **Used:** 60,741,800 B
Max: 263,979,008 B

Total loaded: 13,191 **Shared loaded:** 0
Total unloaded: 568,519 **Shared unloaded:** 0

Live: 82 **Daemon:** 19
Live peak: 82 **Total started:** 141

Ring service profile

I said that this was more about comparing two technology stacks than two programming languages but is that completely true? Perhaps the philosophical differences between the two programming languages had subtle, yet profound, effects on the two different technology stacks.

Martin Odersky is the inventor of Scala. He believes that you can have Functional Programming and Object Oriented Programing on equal footing.

	clojure ring jetty	scala spray akka	scala finatra netty
requests / sec	100.00	13.33	33.33
average latency (ms)	200.00	13.00	11.00
CPU %	96.70	26.50	75.10
write IOPs / sec	346.00	54.80	129.00
framework %	21.40	57.00	40.10
threads	82.00	59.00	62.00
max heap used (MB)	251.75	1,664.00	1,831.00

In FP, you are not supposed to mutate state. Combine that with OOP and you are creating a lot of objects all the time. This puts more pressure on Garbage Collection.

Rich Hickey is the inventor of Clojure which has only enough support of OOP to provide good integration with Java and the ability to create rudimentary, light-weight objects. Instead, Rich emphasizes Persistent Data Structures and Software Transactional Memory to make the highly concurrent stream based processing of server side FP more efficient in the JVM.