Scala 3 slowed us down?

Dec 7, 2025

Is this clickbait? Not really.
Is this the fault of the language or the compiler? Definitely not.
Rather, it was part of a rushed migration. Sharing the lessons learned in the process.

I was refreshing one of our services. Part of this process was to migrate codebase from Scala 2.13 to Scala 3. I've done this a few times before and overall had a positive experience. Well, at least until we talk about projects with macro wizardry.

The service in question had no macros at all, but it was at the heart of data ingestion, so performance was not an afterthought.

I did it as usual - updating dependencies, compiler options and some type/syntax changes.

Then after resolving few tricky implicit resolutions and config derivations, project compiled on Scala 3.7.3 🎉

All tests passed, end-to-end flow locally works perfectly fine, so I decided to roll out the changes in a testing environment. Similarly, no issues at all. No concerning logs, all metrics ranging from infrastructure, through JVM up to application level look healthy.

With that in mind, I began a staged rollout. Again, all seem good. I kept observing the service but it looked like my job is done.

Well, as you probably can guess, it wasn't.

The mysterious slowdown

After 5-6 hours, Kafka lag started increasing on a few environments. Of course, this wasn't something new. Most often it is caused by a spike of data. We have pretty advanced machinery to deal with that. Usually the lag resolves by itself without any manual action.

However, this time *something* was off. Upstream load turned out to be relatively modest, yet we needed much more instances of the service - meaning the processing rate per instance dropped. I was confused to say the least. Why would it decrease the processing rate just on these environments?

Anyway, we decided to rollback the changes - this brought the rate back.

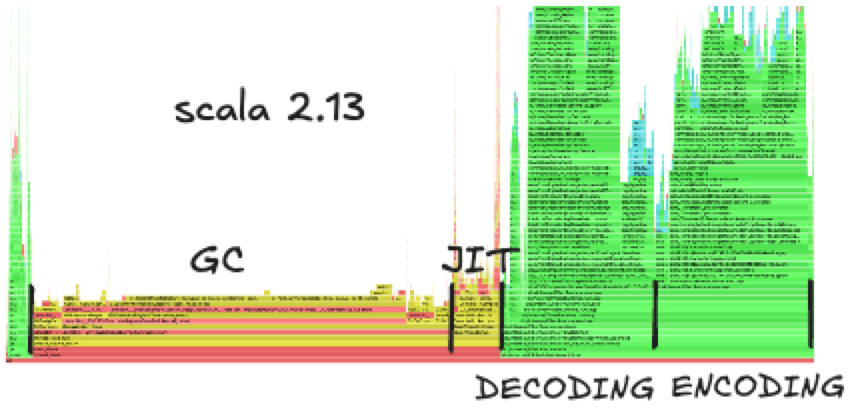
Digging deeper

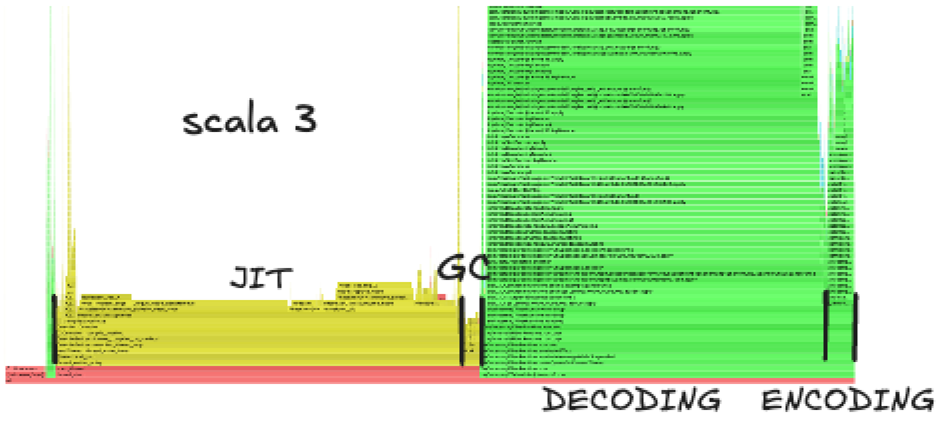
I came back to testing. In particular, load testing. However similarly as on production environments I did not notice regression. So I played around with different payloads and granularity of messages. To my surprise, for more fine-grained, heterogeneous workloads, the processing rate significantly dropped.

Still, I had no idea why it would happen, but my bet was in the dependencies. Therefore, I tried one-by-one, reverting the serialization library, database SDK, base Docker image and even config libraries. None of these made any changes.

This made me pull out the big guns. I profiled the service using async-profiler and indeed

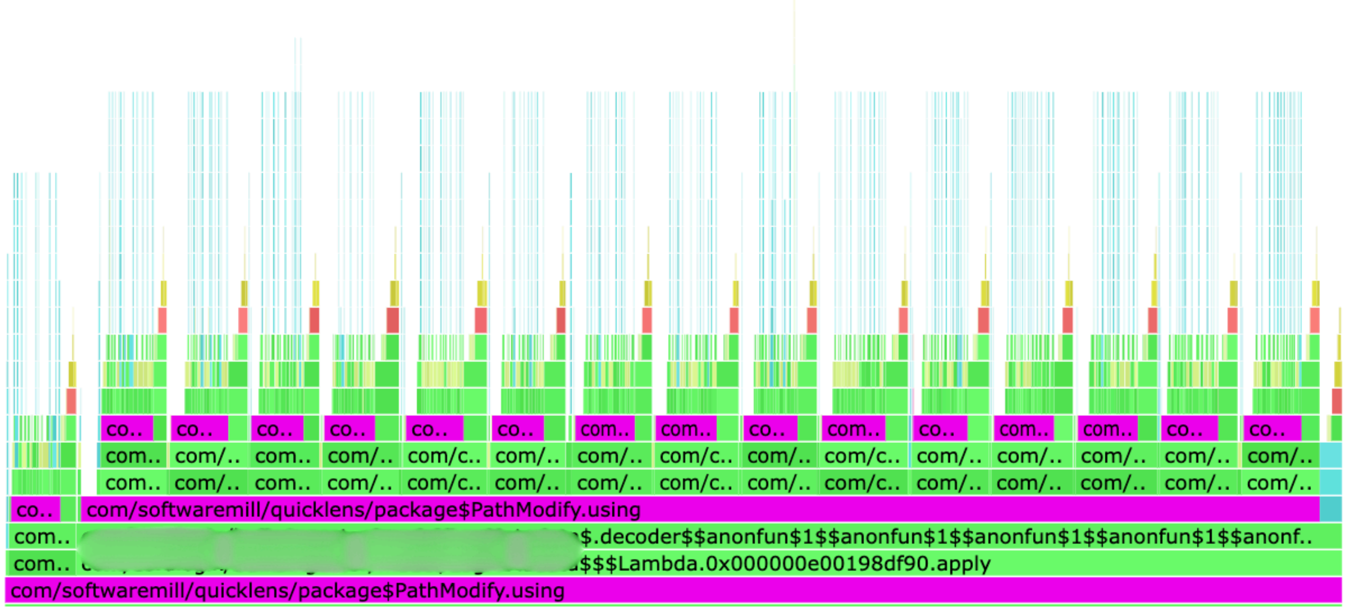
CPU profile looked vastly different on Scala 3 than on 2.13.





JVM-level CPU time was now dominated by JIT compiler while application-level by decoding.

Looking at the top of Scala 3 flamegraph I noticed a long quicklens call.



What used to be transparent (frankly, I didn't even realize we used the library), now took almost half of the total CPU time. I compared how it looks on Scala 2.13 and it was barely noticeable with around 0.5% samples.

Turns out there was indeed a subtle bug making chained evaluations inefficient in Scala 3. This also explained why the JVM spent so much time compiling.

After upgrading the library, performance and CPU characteristics on Scala 3 became indistinguishable from Scala 2.13.

Takeaways

While the details of the bug are pretty interesting(hats off to the SoftwareMill team for catching it!), that's not my point here. I want to emphasize that **libraries can behave very differently between Scala versions**, especially when they rely on meta-programming.

Even if your migration is seamless and the service runs fine on Scala 3 - when performance is not just a nice-to-have, do not assume. Know your hotspots and benchmark them. Otherwise, your code will benchmark you, revealing bottlenecks in places you didn't even know existed.

kmaliszewski.dev [at] gmail.com kmaliszewski9 :)
mrzysztof