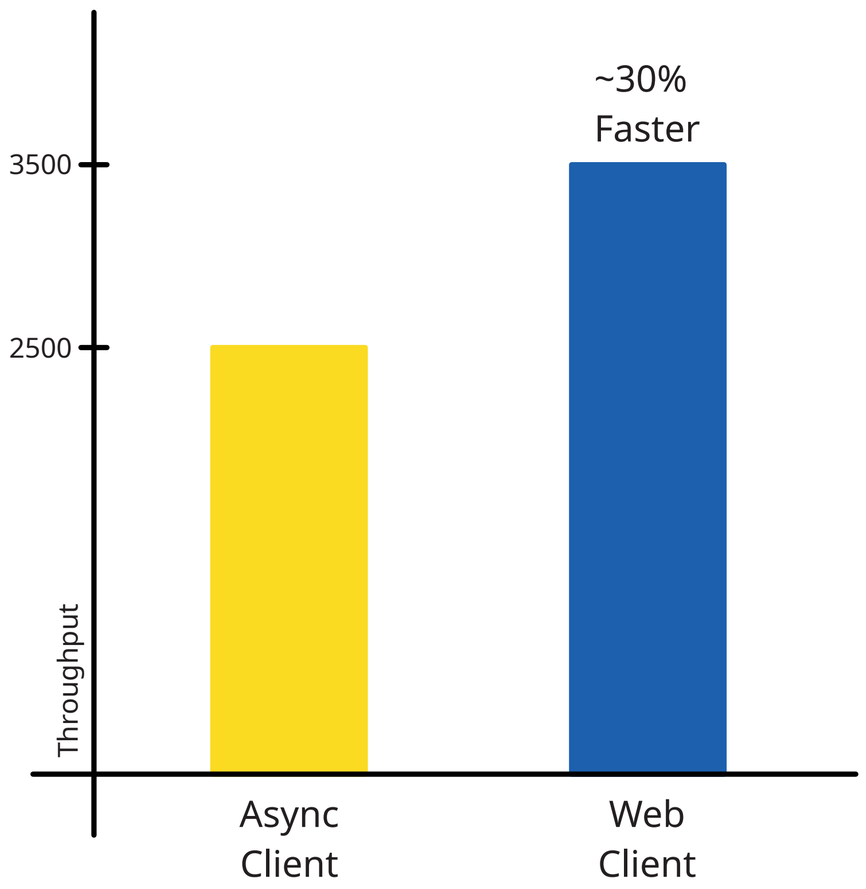
HTTP*Extravaganza:* Less code more speed

A Comparative Evaluation of Java HTTP Clients for Efficient Backend Service Communication

What is a typical irritating **PROBLEM** today? When a website takes too long to load. But what is the reason and why does it differ so much between different websites? Well, it could be caused by several reasons such as your phone or internet connection. However, in the research presented here, in collaboration with Inter IKEA, the focus has been on the server's side of the equation. Specifically, the server's ability to gather data from other servers using an HTTP client. The client is responsible for gathering a lot of data from several places and at the same time be efficient and correct. This client is just a server's version of a web browser, without the fancy graphics!



Now, what **METHOD** to use to implement an HTTP client to test it? For the research conducted both the implementation difficulty and the performance were investigated. You may think that the difficulty of coding something is subjective and that may be true. Although, if you focus on the amount of code itself the subjectivity disappears. What was also evaluated was the performance of three clients and since you do not want to interrupt some actual systems, mockups and local tests were made to imitate the real systems. To answer the question of which client performs better, a comprehensive local test was made and used, as mentioned previously. This simulated things such as several users utilizing the http client's server at the same time. This allowed the research to be done on several usage scenarios.

Looking at the **RESULTS**, it's clear that WebClient's reactive programming reduces the amount of written code. This showed itself in fewer lines of code and in a different coding style with the subjective observation of it being easier to understand. When the performance tests were run, a clear trend started to show itself. The WebClient was faster, regarding both throughput and response times. What was also discovered was that the client that was not asynchronous was much slower in all tests, except when forcing asynchronous behavior with a lot of threads. Then it actually showed promise against the others in performance. Although the many threads were using a lot of computational power.

```
try (InputStream bodyStream =
getBodyStream(httpResponse)) {
  T result = mapper.
readValue(bodyStream,
typeClass);
  return result;
} catch (Exception e) {
  logger.warn("Unable to parse
response {}", e.getMessage());
  throw new
HttpServiceException(ErrorType.
UNABLE_TO_PARSE_RESPONSE);}
```

```
.bodyToMono(returnType)
```

Async
Client

Web
Client

A **DISCUSSION** about the results will clear things up! The greater performance of the WebClient was most likely caused by the reactive programming basis for it. That means that it handles the server's threads differently, via a so-called event-loop. This event-loop gives the WebClient the ability to be just as fast as a normal asynchronous client based on many threads but with much fewer threads. It can do this because each thread can be used to perform more work. But why would the use of more threads hurt performance? That could be because of many reasons; however, the research suggests that the server's handling of too many threads takes a large part of the available computational power.

Can we **CONCLUDE** that the research shows some meaningful results? Yes, we can! It showed a true improvement in using WebClient over the other clients both in terms of implementation and performance.

Liam Ljungerud
&
Erik Stjärnquist