### Compare Spring WebFlux and Spring MVC performance through load test

Spring WebFlux and reactive programming are hot topics recent years.

When do we need to use WebFlux or when do we need replace Spring MVC with WebFlux?. In this article I am trying to answer this question from performance view using data collected in load tests.

### Load Test

JMeter is used to do the load test and concurrency ramp up in 2 seconds.

The server side key code listed as following:

MVC controller:

@RestController

public class LoadTestController {

@GetMapping("/loadtest/{latency}")

public String loadtest(@PathVariable long latency) {

try {

TimeUnit.MILLISECONDS.sleep(latency);

} catch (InterruptedException e) {

return "Error during thread sleep";

}

return "Welcome to MVC world ~";

}

WebFlux controller:

@RestController

public class LoadTestController {

@GetMapping("/loadtest/{latency}")

public Mono<String> loadtest(@PathVariable int latency) {

return Mono.just("Welcome to reactive world 8092 Webflux ~")

.delayElement(Duration.ofMillis(latency));

}

}

The latency value mimics the processing time of application.

#### Test1: WebMVC Load test with 5ms delay

#### Test2: WebFlux Load test with 5ms delay

#### Test3. WebMVC Load test with 200ms delay

#### Test4. WebFlux Load test with 200ms delay

#### Test5. WebMVC 200 max threads and 400 max threads and WebFlux Load test with 10second delay

### Analysis:

#### Saturation is the key of analysis

There are several articles discussing about the comparison of throughput of WebFlux and Spring MVC. Some of them are comparing the absolute value of average response time and throughput..[1,2,3,4]

According to my experiments, the throughput of load test are non-linear. The average response time and throughput in unsaturated area are quite different from the data in saturated area. It makes no sense to compare the WebFlux saturated data with Spring MVC unsaturated data.

First thing to do in load test is to find out the saturation area of the application and normalized value should be used.

In unsaturated area, the average response time keeps constant and it is relatively low. In saturated area the average response time will increase dramatically. The throughput will not increase accordingly when you increase the concurrency. And in deep saturated state we can see the increase of error rate.

#### Unsaturated State

We begin the load test with low concurrency. At this time, system has not reached saturated area and Spring MVC and WebFlux usually behave the similar throughput or average response time.

Fig01 shows when latency is 5ms and concurrency from 100 to 200 for Spring MVC, the average response time almost the same.

Fig02 shows when latency is 5ms and concurrency from 100 to 500 for WebFlux, the average response time almost the same.

Fig03: shows when latency is 200ms and concurrency from 100 to 200 for Spring MVC, the average response time almost the same.

Fig04: shows when latency is 200ms and concurrency from 100 to 500 for WebFlux, the average response time almost the same.

The above result tells that both Webflux and Spring MVC have almost same average response time and throughput. This response time is close to the latency time we put in the code.

The reason behind this is Spring MVC and WebFlux are both capable to handle all requests at low concurrency.

#### Near Saturated State

One interesting result is : when approaching the saturated area, Webflux has higher tolerance margin before get into “deep saturated”. For example in Fig.02 and Fig.04, WebFlux is approaching the saturated area when concurrency is close to 500. In Fig.01 ad Fig.03, Spring MVC is approaching the saturated area when concurrency is close to 200.

This could be explained when look into the implementation of request processing. Spring MVC uses thread pool and assign one thread for each request. When the thread pool is full, the new thread will be put into the waiting queue. Once the request is in the waiting queue and main part of the response time is actually the waiting time not the process time.

WebFlux uses different mechanism, it uses NIO and direct memory and one thread to do the multiplexing and assign the requests to different handlers. So it does not have the max thread number limit in thread pool and has no threads switching overhead. In WebFlux system resources are used more utilized than Spring MVC.

A typical example of WebFlux application is Spring Cloud Gateway using OIDC. The application mainly does the authentication and authorization of the requests and routes the requests to different microservices using token relay and it even does not have database operations in main processing stream.

So Netty in WebFlux gains high efficiency based on that fact that most threads are waiting for IO when the processing time is quick. But when the processing time or the response body are really very big, both Spring MVC and WebFlux could get into saturated state.

#### Saturated State

Fig.01 shows when latency is 5ms and concurrency for Spring MVC is 600 the average response time increased dramatically.

Fig.02 shows when latency is 5ms and concurrency for WebFlux is 900 the average response time increased dramatically.

Fig.03 shows when latency is 200ms and concurrency for Spring MVC is 900 the average response time increased dramatically.

Fig.04 shows when latency is 200ms and concurrency for WebFlux is 1500 the average response time increased dramatically.

For similar concurrency conditions Spring MVC is getting into saturated area earlier and WebFlux has higher absolute throughput value than Spring MVC before or near saturated point.

### When to use WebFlux

#### Price for WebFlux

First of all, reactive programming is quite different from synchronous programming like Spring MVC, the learning curve is steep. Also it is difficult to debug , stack trace is useless and need the whole flow to be asynchronous, otherwise you will just block the event loop. And not all databases support asynchronous database driver.[5]

#### Use WebFlux

If your current Spring MVC based application is running fine, that probably means your application is not in saturated area, so do not need switch to WebFlux.

If your system is already in Saturated state, we can consider switch to WebFlux. Fig.05 shows even when the latency is as high as 10seconds, WebFlux has lower response time compared with Spring MVC while Spring MVC is getting in saturated.

If your system is near saturated state, another option which is to increase the max thread number of the thread pool. The gray line in Fig.05 shows the average response time is reduce like 50% when max-thread number is increased from 200 to 400.

There might be scenario that the application load is really heavy and even if you switch to WebFlux will not help. Then you have to upgrade the hardware or consider using clustered or distributed system and we will discuss this in the future.

### Conclusion:

When you consider use or switch from Spring MVC to WebFlux, whether current system in saturation state is the key factor. If your system is or getting into the saturation area, WebFlux might be an good option to increase the throughput and reduce average response time. Another quick option to try is to increase the max thread number of thread pool in Spring MVC.

References:

1.https://thepracticaldeveloper.com/full-reactive-stack-4-conclusions/#clients-side-average-time-per-request

2.<https://dev.to/azure/spring-boot-performance-benchmarks-with-tomcat-undertow-and-webflux-4d8k>

3. <https://dzone.com/articles/raw-performance-numbers-spring-boot-2-webflux-vs-s>

4.<https://medium.com/@filia.aleks/microservice-performance-battle-spring-mvc-vs-webflux-80d39fd81bf0>

5. <https://stackoverflow.com/questions/62765757/springdoc-webmvc-vs-webflux>