Subject: Re: Book idea: U++ Web development
Posted by BetoValle on Mon, 18 May 2026 15:26:41 GMT
View Forum Message <> Reply to Message

Você pode colocar isso de forma muito inteligente e técnica, sem transformar em “guerra de frameworks”. O ponto forte aqui não é simplesmente throughput bruto — porque os payloads parecem diferentes (`Avg Bytes` muito diferente) — mas sim:

* estabilidade;
* zero errors;
* latência menor;
* consistência;
* servidor nativo;
* baixo consumo;
* simplicidade de deploy.

E principalmente:

* depois da correção thread-safe, o Skylark suportou carga sustentada sem falhas.

Isso é o mais importante.

I also performed some comparative stress testing using JMeter remotely against the same VPS environment.

Java/Tomcat test:

* 300 threads
* 2 minutes duration

Results:

```text id="tb8j8z"
Samples:      216,749
Average:      155 ms
Min:          8 ms
Max:          31,605 ms
Std Dev:      423.60
Errors:       0.781%
Throughput:   1790 req/sec
```

U++ / Skylark native server test:

* 300 threads
* 5 minutes duration

Results:

```text id="pkm30h"
Samples:      222,208
Average:      132 ms
Min:          6 ms
Max:          7,318 ms
Std Dev:      359.62
Errors:       0.000%
Throughput:   736 req/sec
```

Of course, these numbers should not be interpreted as a simplistic "framework A vs framework B" comparison, because payload size and implementation details were different.

However, some very important practical observations became clear:

* Skylark maintained lower average latency
* maximum latency spikes were dramatically lower
* zero errors occurred during the test
* the native executable remained stable for a long sustained period
* memory/resource consumption stayed very small
* deployment complexity was minimal compared to a traditional Java stack

The most important point for me was not peak throughput itself, but production stability after redesigning the application to avoid shared/global SQL session state.

After implementing request-local database handling and deterministic object destruction (RAII), the native server became extremely reliable.

I think practical benchmark discussions like this would add enormous value to the book, especially when combined with explanations about:

* threading models
* shared state
* request isolation
* object lifetime
* production architecture decisions

because these are the areas where native web servers behave very differently from managed runtimes like Java or .NET.

---