

CONSIDERATIONS FOR INTEGRATING VIRTUAL THREADS IN A JAVA FRAMEWORK: A QUARKUS EXAMPLE IN A RESOURCE-CONSTRAINED ENVI- RONMENT

A. NAVARRO^{1,2} J. PONGE^{1,2} F. LE MOUËL² C. ESCOFFIER¹

¹RED HAT ²CITI LABORATORY-INSA LYON

2023

1 Context

2 Integration

3 The experiment

4 Experimental protocol

5 Results

THE END OF THE MONOLITH

- difficult to maintain
- dependency hell
- reboot for every change
- sub-optimal deployment
- limit scalability
- technology lock-in

from

Microservices: yesterday, today, and tomorrow - Dragoni & al

MICROSERVICES

- *difficult to maintain*
- split in smaller entities
- *dependency hell*
- less dependencies per service
- *reboot for every change*
- reboot only impacted service
- *sub-optimal deployment*
- per-service deployment
- *limits scalability*
- scale each service
- *technology lock-in*
- per service technology

Remark

Transitioning to cloud becomes a thing

MAKE MICROSERVICES CHEAP

resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency

resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency

resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency
resource-efficiency

Problem

communications over the network are unreliable and slow

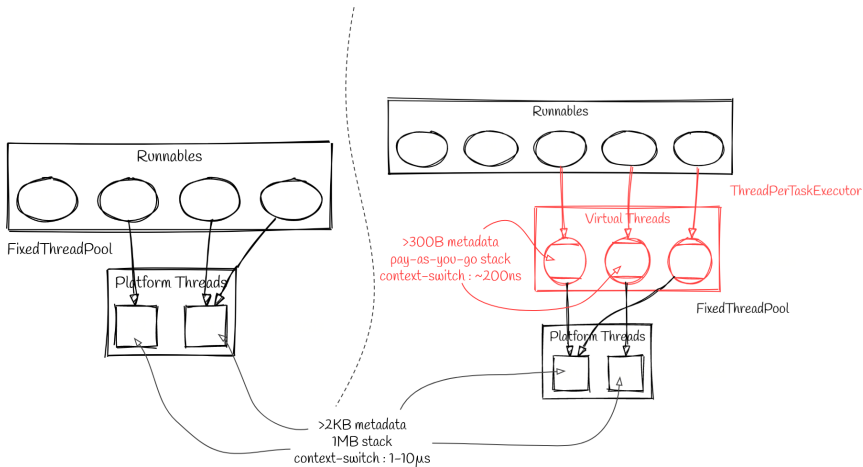
Solutions

efficient and timely communication between services historically brought by non-blocking, asynchronous abstractions in the OS

NON-BLOCKING ABSTRACTIONS

- asynchronous callbacks
- promises and futures
- reactive streams
- coroutines
- async functions
- light threads

FINALLY, VIRTUAL THREADS



CODE COMPARISON - BLOCKING

```
var names = getAll();
var quotes = getQuotes(names.size());
for(int i=0; i < names.size();i++){
    names.get(i).surname+= "- "+quotes.get(i);
}
return names;
```

CODE COMPARISON - ASYNCHRONOUS CALLBACKS

```
getAll( names => {  
    getQuotes(names.size(), quotes => {  
        for(int i=0; i < names.size();i ++){  
            names.get(i).surname+= "- "+quotes.get(i);  
        }  
        //continuation  
    })  
});
```

CODE COMPARISON - REACTIVE STREAMS

```
var names = getAll().memoize().indefinitely();
var quotes = names.onItem().transformToUni(list ->
    getQuotes(list.size()));
return Uni.combine().all()
    .unis(names,quotes).asTuple()
    .onItem().transform(tuple -> {
        var nList=tuple.getItem1();
        //can await it since it is already resolved
        var qList = tuple.getItem2();
        for(int i=0; i < namesList.size();i++){
            nList.get(i).surname += " - "+qList.get(i);
        }
        return namesList;
    });
```

CODE COMPARISON - VIRTUAL THREADS

```
var names = getAll();  
var quotes = getQuotes(names.size());  
for(int i=0; i < names.size();i++){  
    names.get(i).surname+= "- "+quotes.get(i);  
}  
return names;
```

*We should do (as wise programmers aware of our limitations) our utmost best to shorten the conceptual gap between the static program and the dynamic process, to **make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.***

Edgar. J. Dijkstra, 1968

TABLE OF CONTENTS

1 Context

2 Integration

3 The experiment

4 Experimental protocol

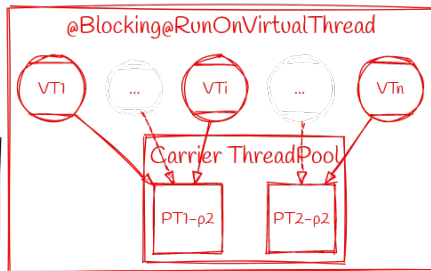
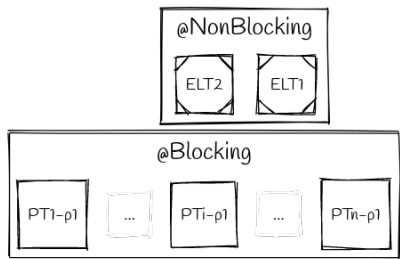
5 Results

THREE INTEGRATION STRATEGIES

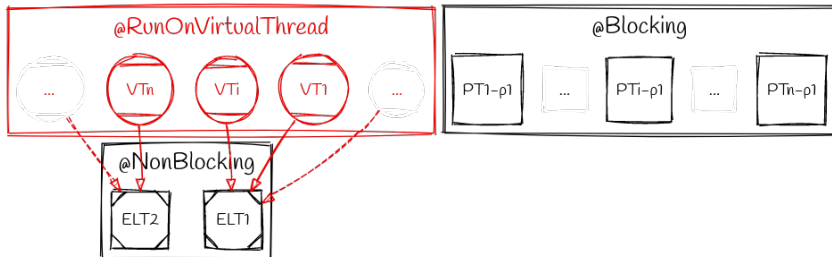
Strategy	Pros	Cons
Forking worker model	Simple, fits virtual threads model	Context switches
Using event-loop as carrier	No context-switch, Fewer threads overall	Potential deadlocks
Modifying Netty event-loops to be virtual threads	Integration done at the Netty level, Netty-based frameworks would benefit from it	Can't modify Netty upstream, unpredictable effects

Table: Comparison of the different Quarkus-virtual-threads options

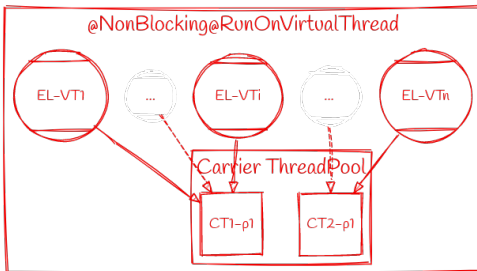
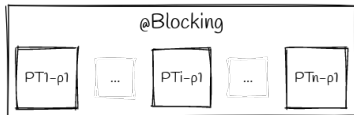
FORKING THE WORKER MODEL



REUSING EVENT-LOOPS



"VIRTUALIZING" NETTY EVENT-LOOPS



DEADLOCK: ROLES OF THE EVENT-LOOP

1. event-loop for the entire non-blocking inner-workings of the framework,
2. event-loop for the non-blocking endpoint handlers,
3. work as a carrier thread for virtual threads

DEADLOCK SITUATION

SCHEMA

Conclusion

The event-loop can't reuse locks *as a carrier*

INTEGRATION STRATEGIES SUMMARY

Strategy	Pros	Cons
Forking worker model	Simple, fits virtual threads model	Context switches
Using event-loop as carrier	No context-switch, Fewer threads overall	Potential deadlocks
Modifying Netty event-loops to be virtual threads	Integration done at the Netty level, Netty-based frameworks would benefit from it	Can't modify Netty upstream, unpredictable effects

Table: Comparison of the different Quarkus-virtual-threads options

FINAL CHOICE: FORKING THE WORKER MODEL

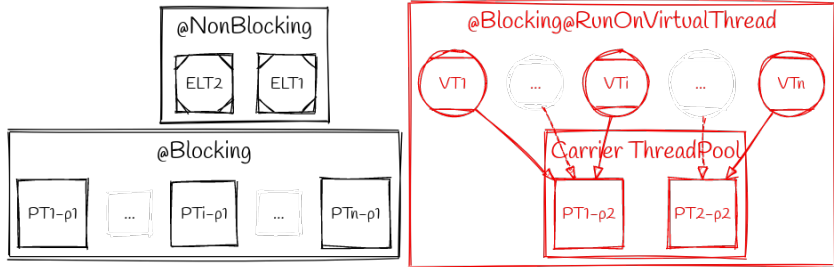


TABLE OF CONTENTS

1 Context

2 Integration

3 The experiment

4 Experimental protocol

5 Results

THE EXPERIMENT

Goal

Measure how performance of the application is affected by replacing reactive endpoints with virtual-threads offloading, in conditions close to

Hypothesis

Quarkus-virtual-threads should perform better than **Quarkus-blocking** but not as well as **Quarkus-reactive**

Limited resources

- 512MB memory
 - 0.5 vCPU
 - 256MB heap
- limited resources

TABLE OF CONTENTS

- 1 Context
- 2 Integration
- 3 The experiment
- 4 Experimental protocol**
- 5 Results

ELEMENTS OF THE EXPERIMENT

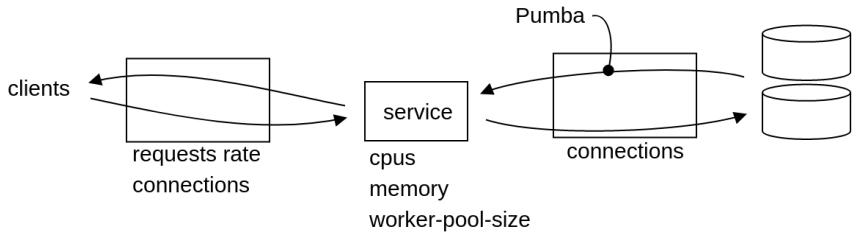


TABLE OF CONTENTS

1 Context

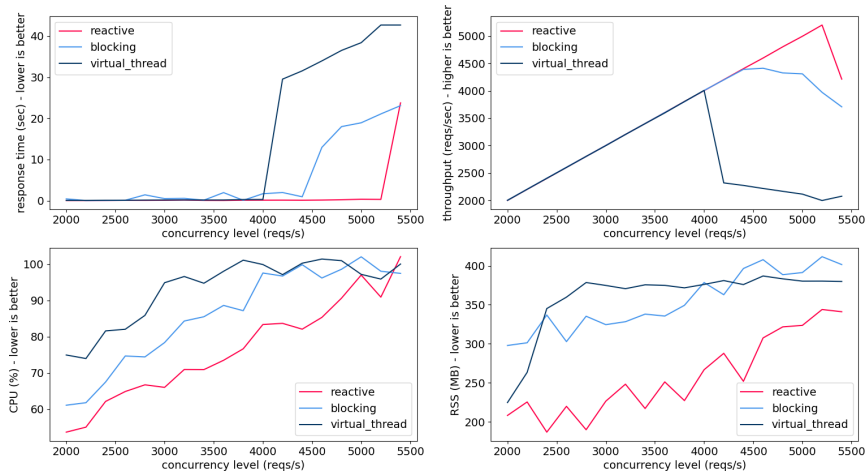
2 Integration

3 The experiment

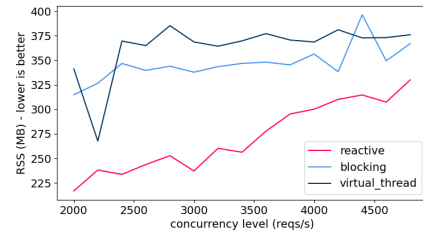
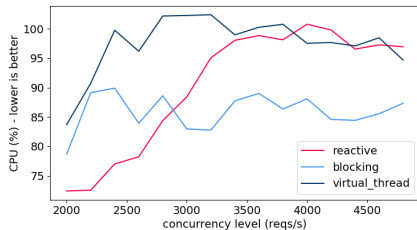
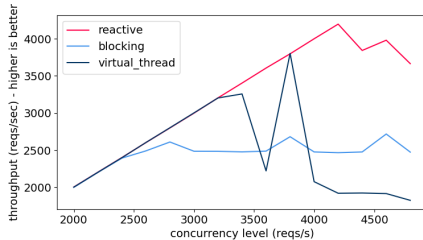
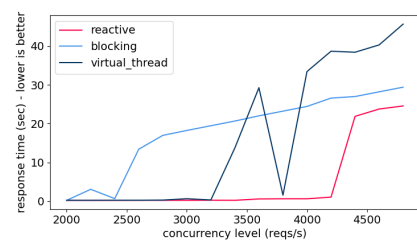
4 Experimental protocol

5 Results

RESULTS WITHOUT DELAY



RESULTS WITH 200MS DELAY



CONCLUSIONS

performance	Quarkus-blocking < Quarkus-virtual-threads < C
resource efficiency	x