Considerations for integrating virtual threads in a Java framework: a Quarkus example in a resource-constrained environment

A. Navarro[1,2] J. Ponge[1,2] F. Le Mouël[2] C. Escoffier[1]

[1]Red Hat [2]CITI 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
resource-efficiency
resource-efficiency
resource-efficiency

Scaling and reactive programming

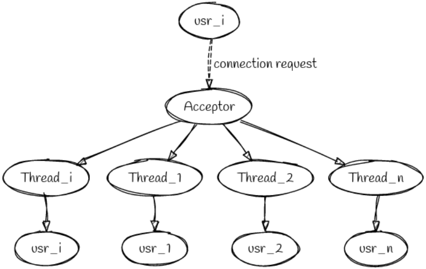
Problem

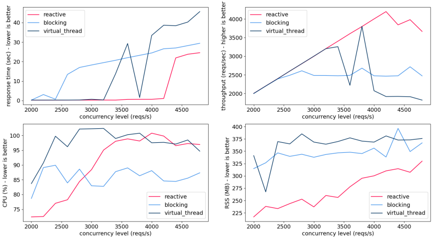
Communications over the network are unreliable and slow.

Goal

Achieving efficient and timely communication between services.

Non-blocking and blocking programming





Non-blocking abstractions

- asynchronous callbacks
- promises and futures
- reactive streams

- coroutines
- async functions
- light 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;
```

Asynchronous programming abstractions

> *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

Finally, virtual threads

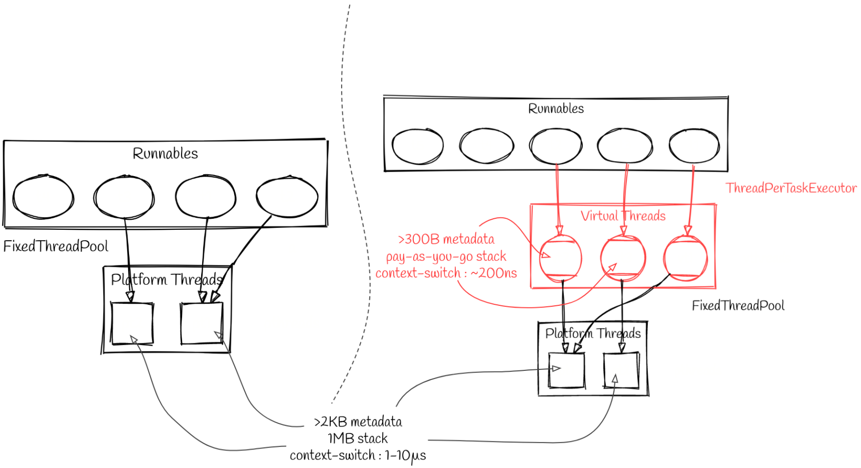


Table of Contents

1 Context

2 Integration

3 The experiment

4 Experimental protocol

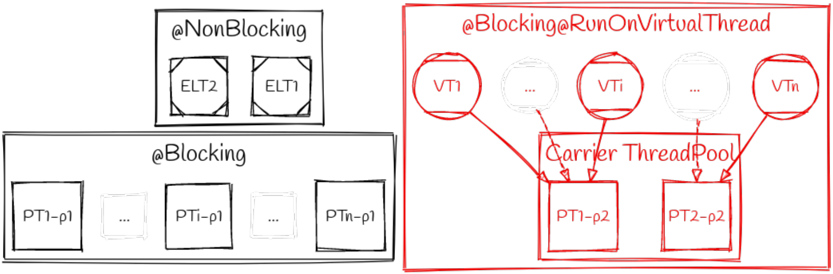
5 Results

Three integration strategies

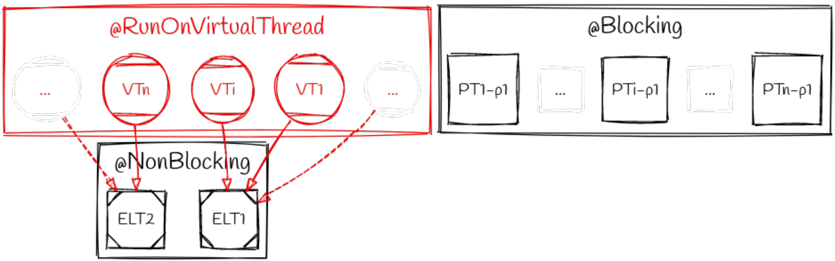
Strategy	Pros	Cons
Forking worker model	Simple, fits virtual threads model	Context switches
Using event-loop as carrier	No context-switch, Fewer threads overall	Potential deadlocks
Modifying Netty event-loops to be virtual threads	Integration done at the Netty level, Netty-based frameworks would benefit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

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

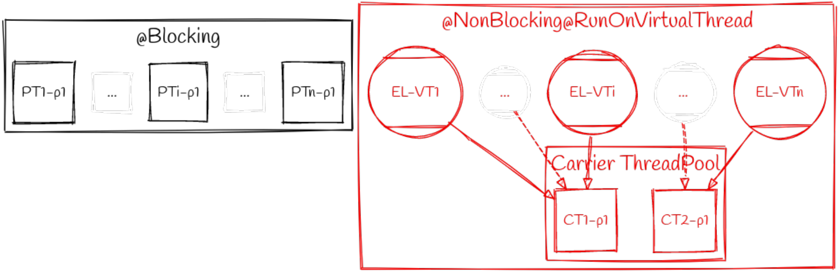
Forking the worker model



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

Deadlock: sharing locks

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

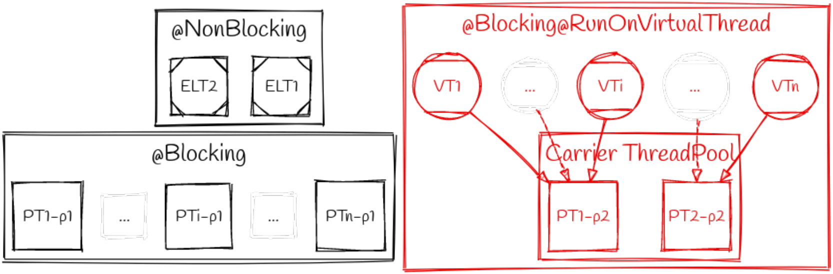


Table of Contents

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**

The environment

Limited resouces

- 512MB memory
- 0.5 vCPU
- 256MB heap

- limited resouces

Table of Contents

1 Context

2 Integration

3 The experiment

4 Experimental protocol

5 Results

Elements of the experiment

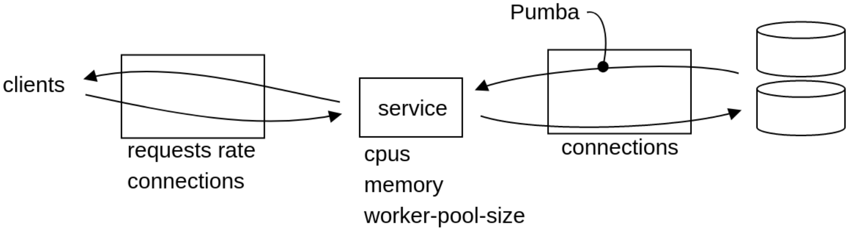
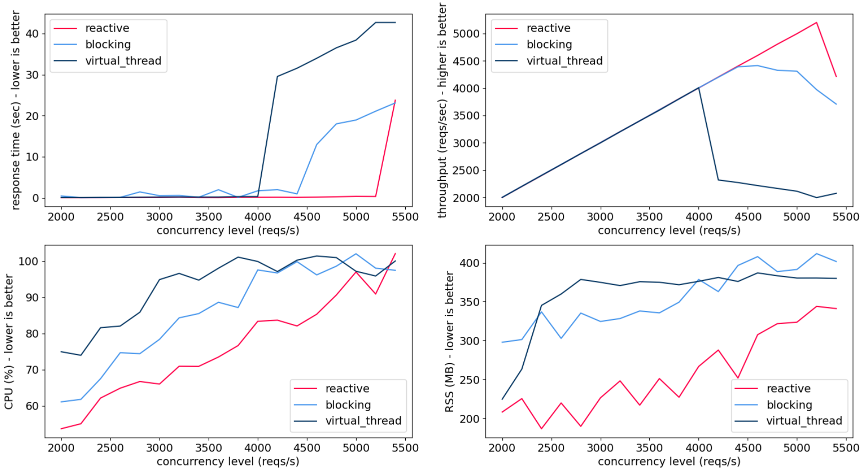


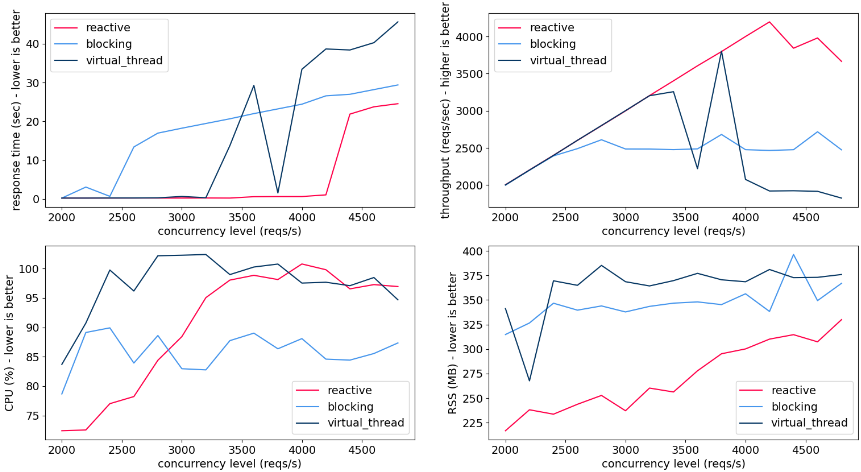
Table of Contents

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 <
resource efficiency	x