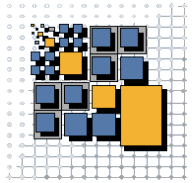
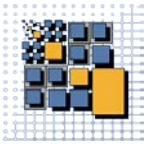


Cold Start Influencing Factors in Function as a Service

Johannes Manner · Martin Endreß · Tobias Heckel · Guido Wirtz
johannes.manner@uni-bamberg.de

Distributed Systems Group
Otto-Friedrich-University Bamberg, Germany

Agenda



Research Questions

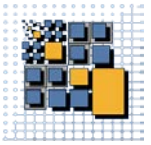
- How to benchmark cold starts of cloud functions consistently to get repeatable experiments and results?
- Which factors written down as hypotheses influence the cold start of cloud functions?

Why Investigating Cold Starts?

- In General: First Execution on a newly allocated VM, container etc. is always slower
- Challenges:
  Performance Unpredictability (No.5)
  Scaling Quickly (No.8) *[Armbrust 2010]*
- Cloud Functions are executed in containerized environments

- Overall question: How much overhead does a “simple” cold start face?
- Which factors need deeper insights?

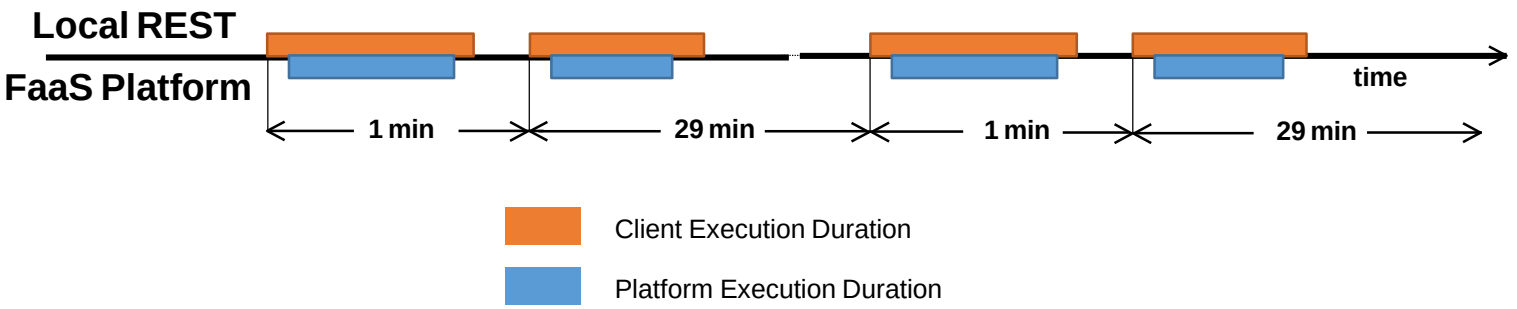
Hypotheses



- H1: Programming Language
- H2: Deployment Package Size
- H3: Memory/CPU Setting
- H4: Number of Dependencies
- H5: Concurrency Level
- H6: Prior Executions
- H7: Container Shutdown

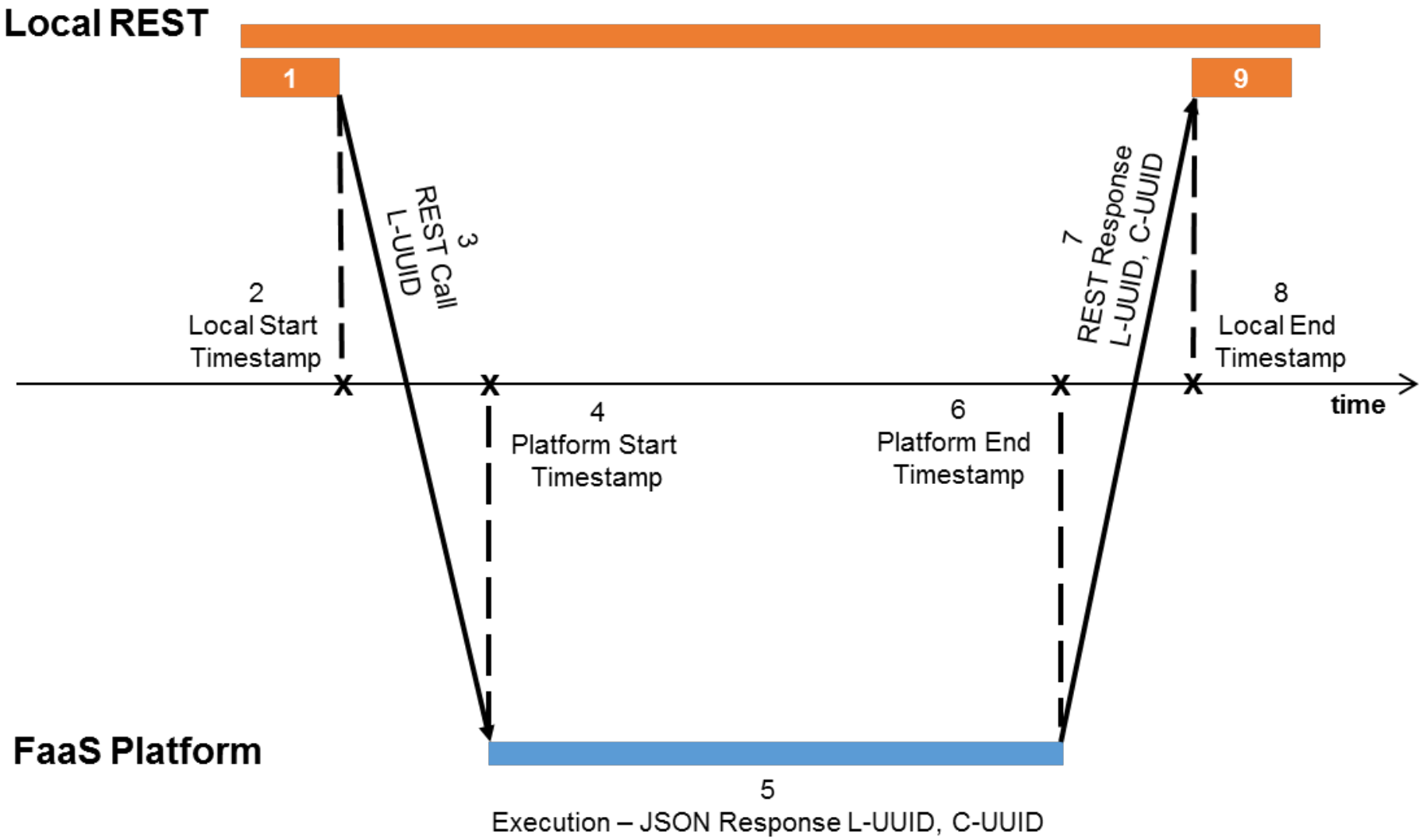
- HX: Discussion during breaks :)

Experiment – Multiple Executions

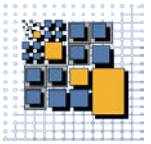


- Measurement of the user perception – the user's cold start
- Network, platform routing etc. is equal for cold and warm starts
  → We only measure the overhead of cold starts
- Having cold and warm executions pairwise since platform shuts down function containers after 20 to 25 minutes (own investigation)

Experiment – Single Execution



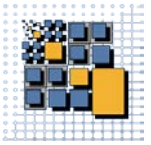
Experiment – Data Dimensions



- Experiment was conducted between 6/25/2018 and 7/1/2018
- Each cloud function was invoked 550 times
- 90 deployed functions – 49500 total executions

	(AWS, Azure)
H1: Programming Language	**(Java, JS)**
H2: Deployment Package Size	**(0, 3, 6, 12, 25, 50, 100, 200, 400 MB)**
H3: Memory/CPU Setting	**(128, 256, 512, 1024, 2048, 3008 MB)**

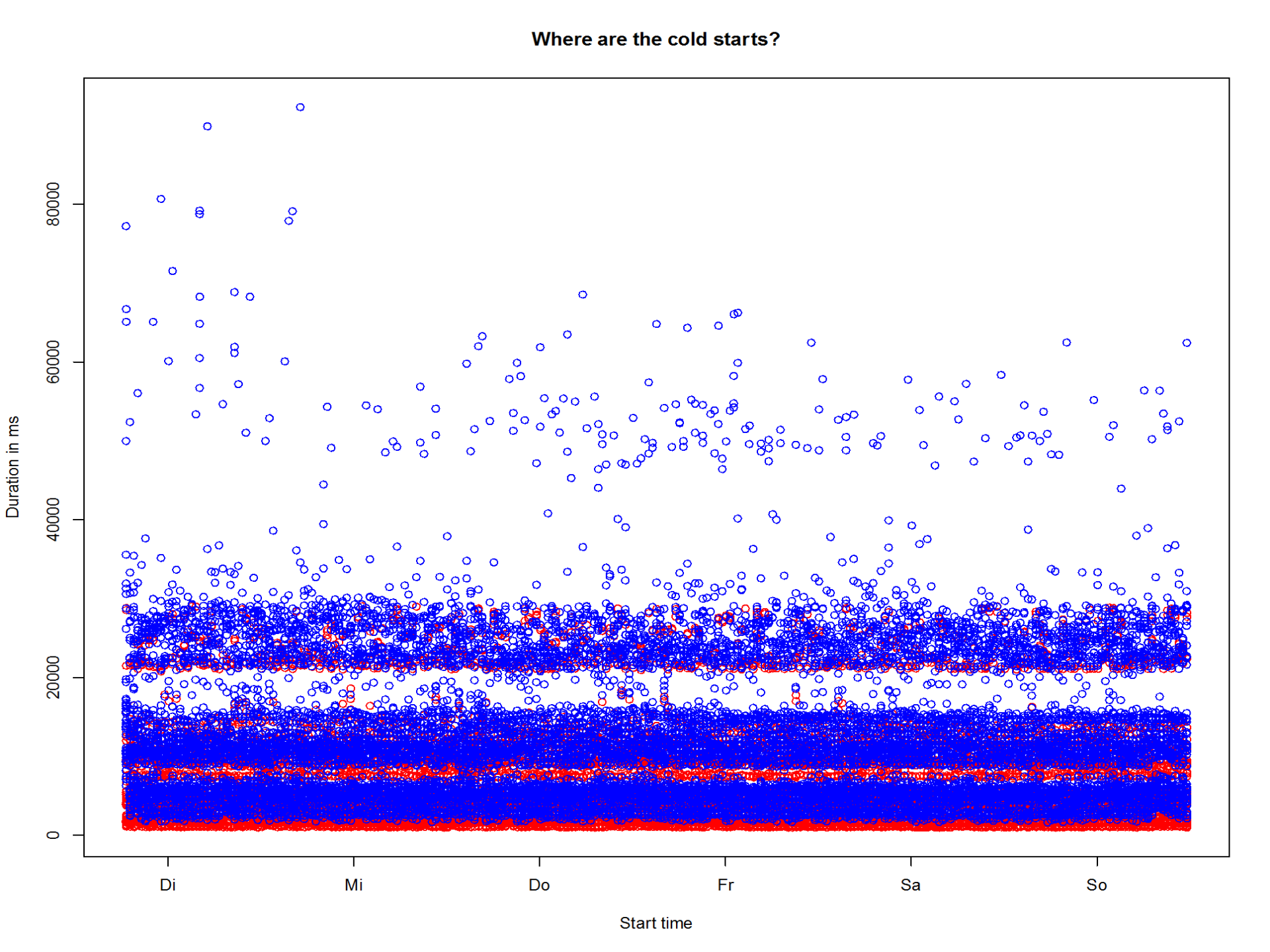
Experiment – Selected Algorithm



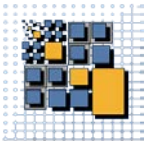
- Recursive Fibonacci

```
long fibonacci(long n) {
  if (n <= 1) {
    return 1;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
```

- Low memory usage $O(n)$
- High CPU usage $O(2^n)$
- Predictable execution time


Where are the cold starts?
Duration in ms
0
20000
40000
60000
80000
Di
Mi
Do
Fr
Sa
So
Start time

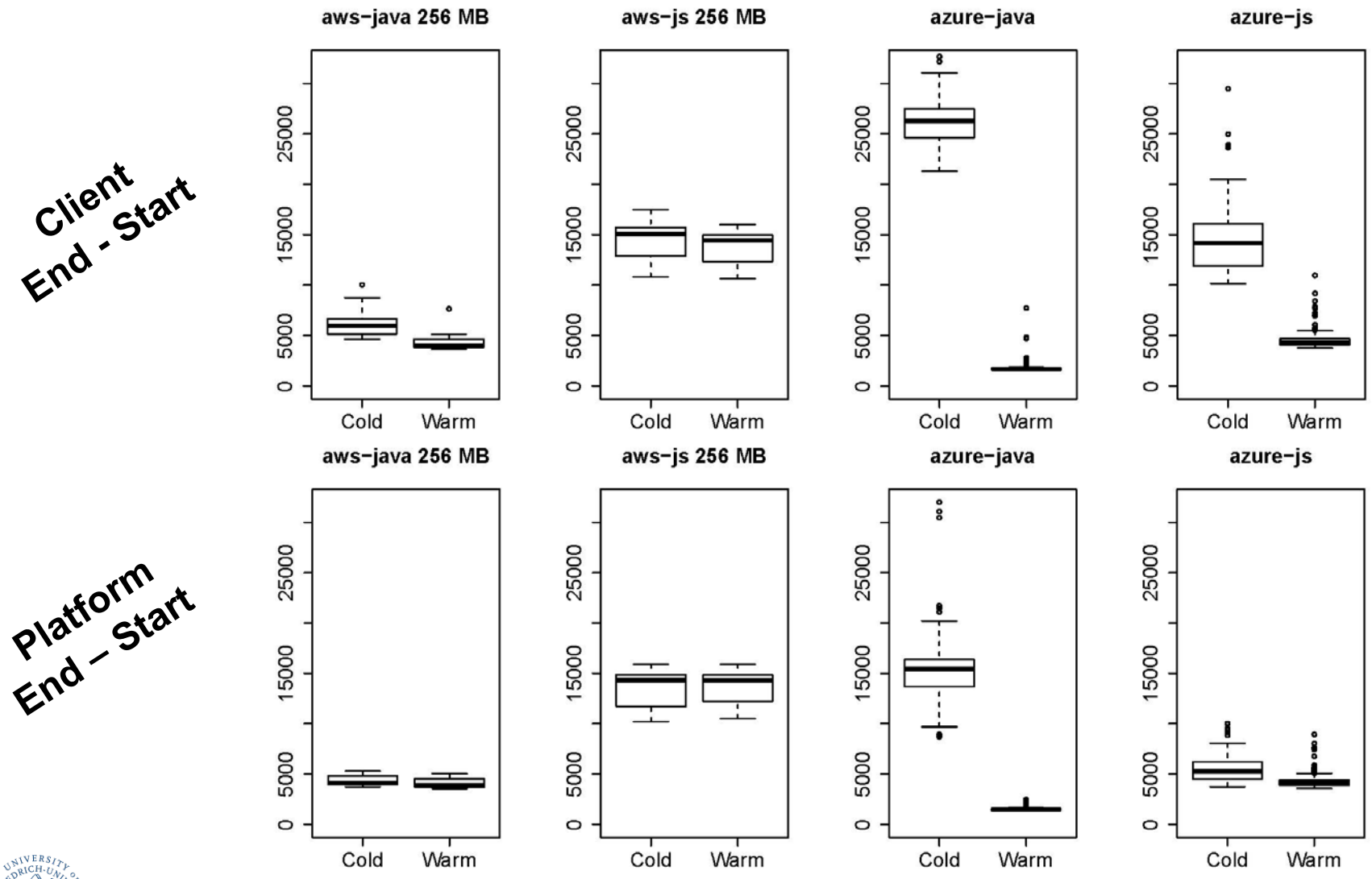
Results – Hypotheses Independent



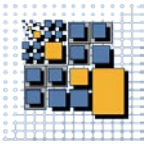
All numbers are mean average values for the execution time in ms

			Cold	Warm	Cold–Warm	
		Client	5961	4211	1750	
	Java	Platform	4329	4082	247	**16%**
	JS	Client	14320	13676	644	
AWS		Platform	13496	13539	-43	**63%**
Azure		Client	26681	1809	24872	
	Java	Platform	15261	1545	13716	**0%**
	JS	Client	14369	4547	9822	
		Platform	5492	4270	1222	**4%**

Results – Hypotheses Independent



Results – Hypotheses Dependent



- H1: Programming Language
  - Table shows AWS values
  - Azure ratio: 2.53
  - Confirm hypothesis based on the ratios

Memory in MB	128	256	512	1024	2048	3008
Java	1980	1750	1292	1113	1257	861
JS	587	644	614	368	589	371
Ratio	3.38	2.72	2.10	3.03	2.14	2.32

- H2: Deployment Package Size
  - Weak, but present correlation
    → Confirm hypothesis except for Azure Java (no clear tendency)

		ρ	$Linear Model$
AWS	Java	0.29	$1510ms + 9\frac{ms}{MB}$
	JS	0.37	$613ms + 12\frac{ms}{MB}$
Azure	Java	-0.15	$25580ms - 7\frac{ms}{MB}$
	JS	0.46	$8571ms + 43\frac{ms}{MB}$

- H3: Memory/ CPU Setting
  - Only quantifiable for AWS
  - Confirm hypothesis

	ρ	$Linear Model$
Java	-0.59	$1634ms - 0.25\frac{ms}{MB}$
JS	-0.20	$606ms - 0.07\frac{ms}{MB}$

Discussion

User point of view

Assumption: AWS bills only for function execution

Parameter setting is use case dependent

$\frac{compiled}{interpreted} : 2 \; to \; 3$

Available Metrics

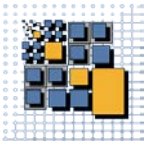
Platform Limitations

Avoid ping strategies – Scaling

Sample Size

Temporal Relevance

Next Steps



- Extending the presented work by including new hypotheses, platforms and dimensions
- Price Analysis based on different load settings (constant load, bursty workloads etc.) and other influential factors. Comparison with other cost benchmarks. Also including the pricing of other ecosystem backend services like databases, notification systems etc.
- Simulation framework for FaaS users to find the best setting for their cloud functions.

I‘m very interested to discuss the presented work and the planned research objects to get your feedback during the poster session ☺

Contact

Johannes Manner

/johannes-manner

johannes.manner@uni-bamberg.de