Master Thesis
Politehnica University Timișoara

Benchmark Evaluation of Scale-Up and Scale-Out Techniques on a Functional Programming Based Microservice

SUPERVISOR:

LECT.DR.ENG. ALEXANDRU TOPÎRCEANU

CANDIDATE:

ARNOLD ATTILA HIGYED

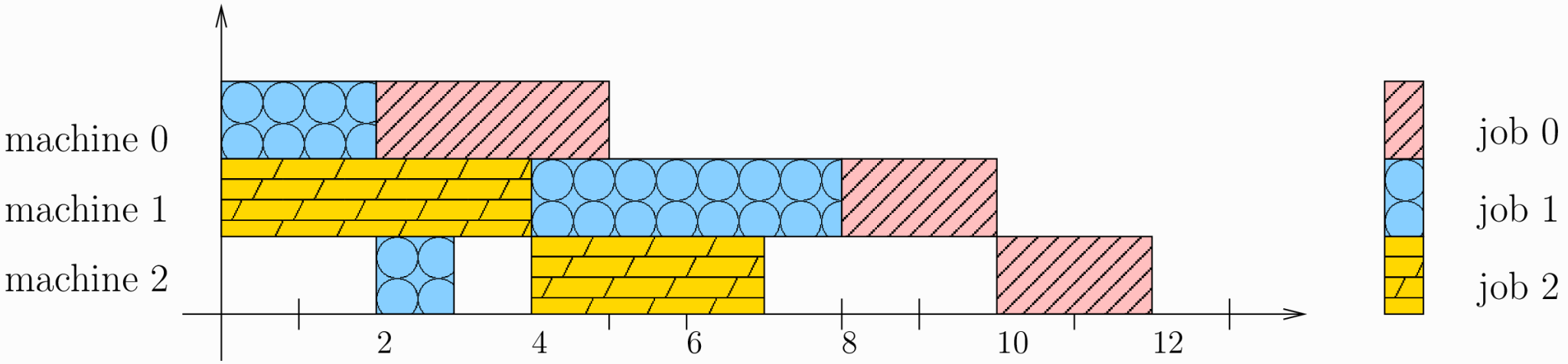
TIMIȘOARA
JUNE 2020

Content

- Introduction
- About the microservice
- Scaling
- Scenarios
- Results
- Conclusions

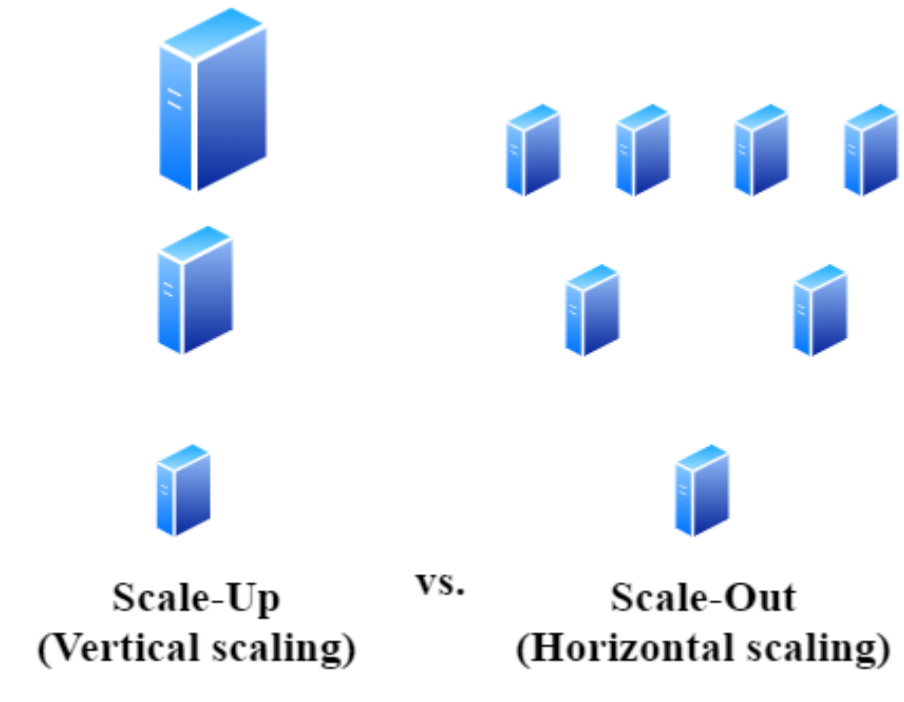
Introduction

What is measured?



Introduction

How it is measured?

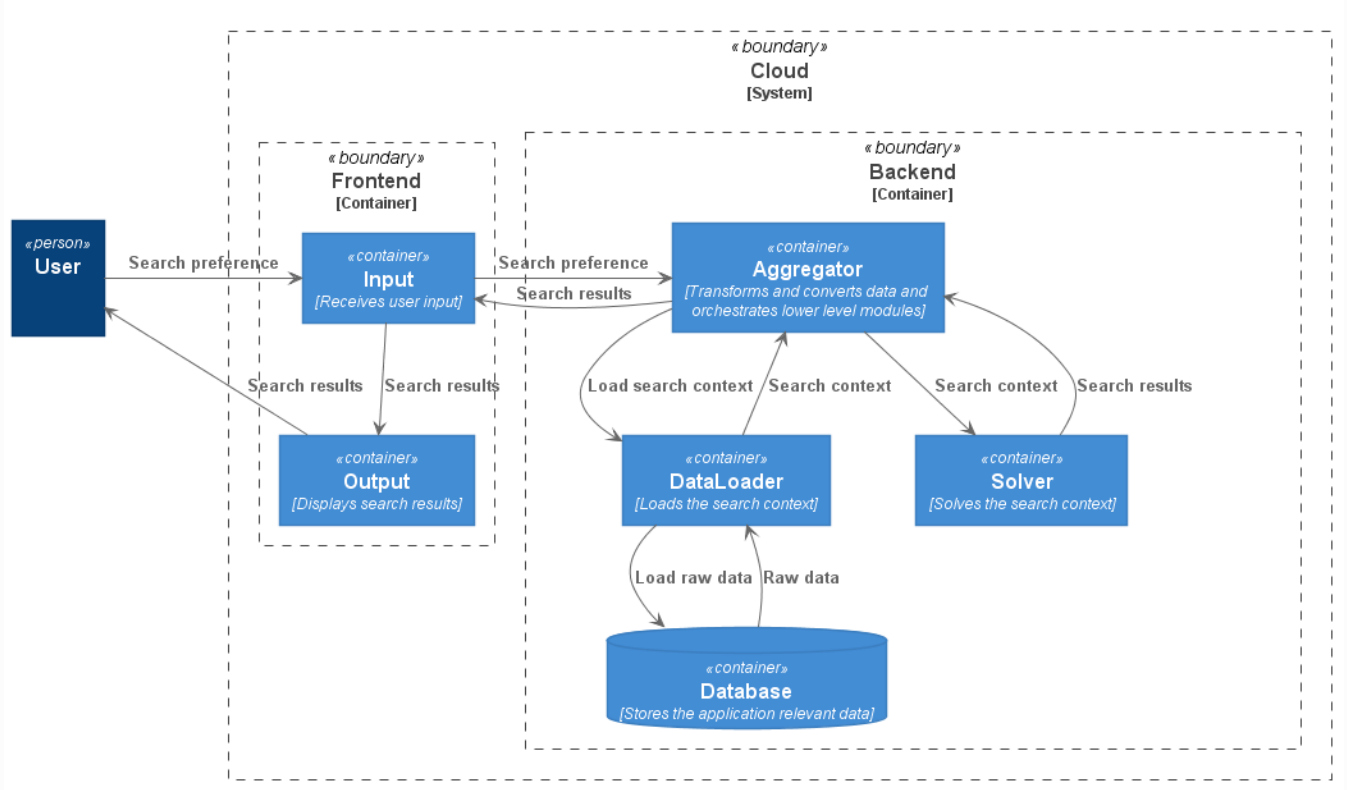


Introduction

- Which scaling technique is better and more load-tolerant?
- How do the two strategies perform together?

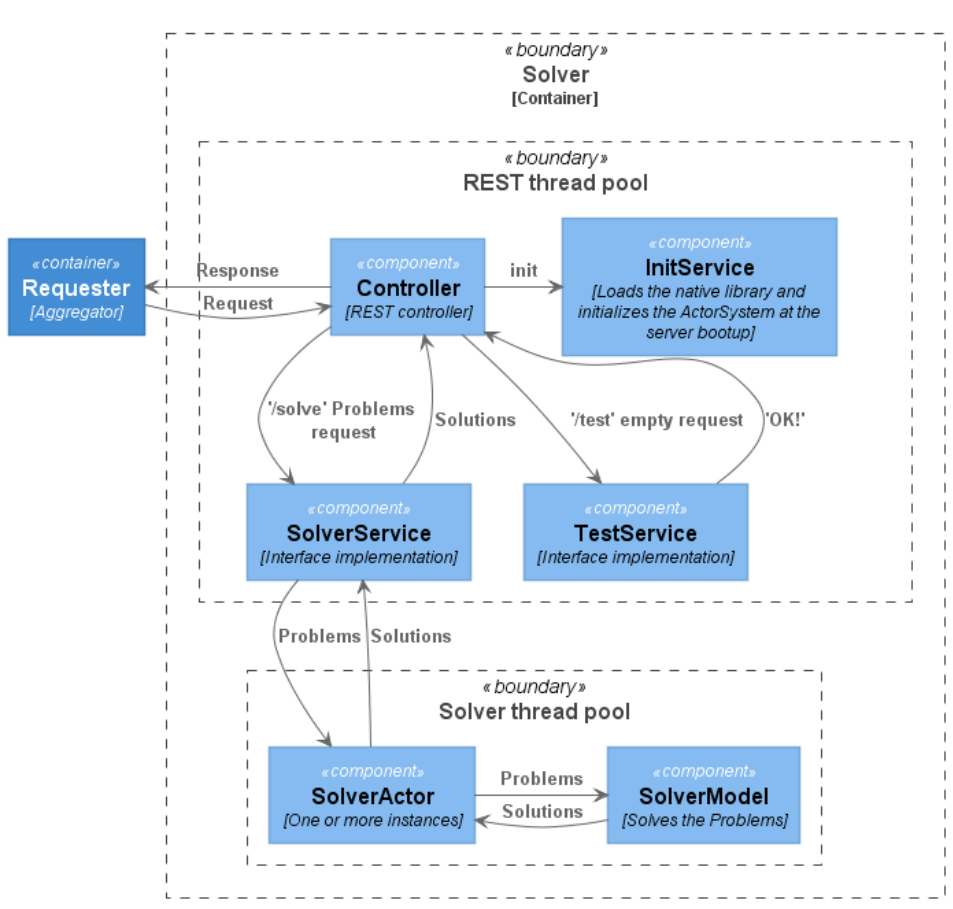
About the microservice

Planning application architecture



About the microservice

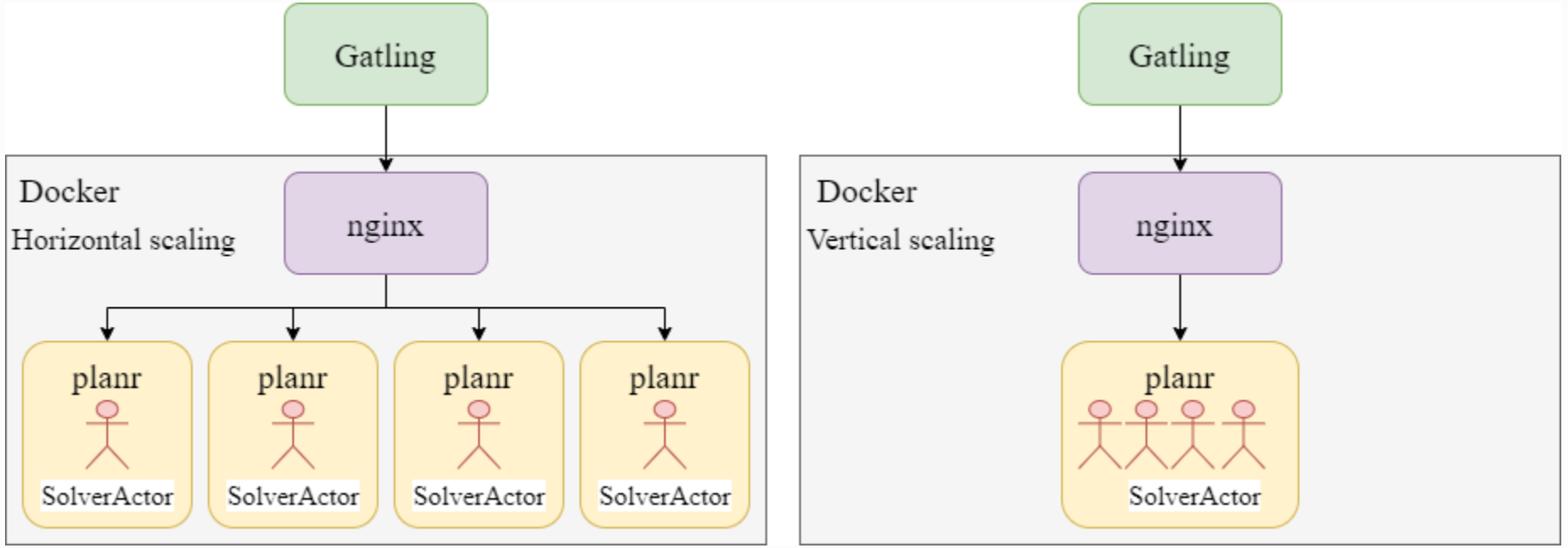
Solver architecture



About the microservice

- Constraints programming
- 6 constraints: operationGrid, sameResource, enforceTimeInterval, operationsRelation, program and disjunctive
- 3 costs: asSoonAsPossible, asTightAsPossible, prefferedTimeInterval

Scaling

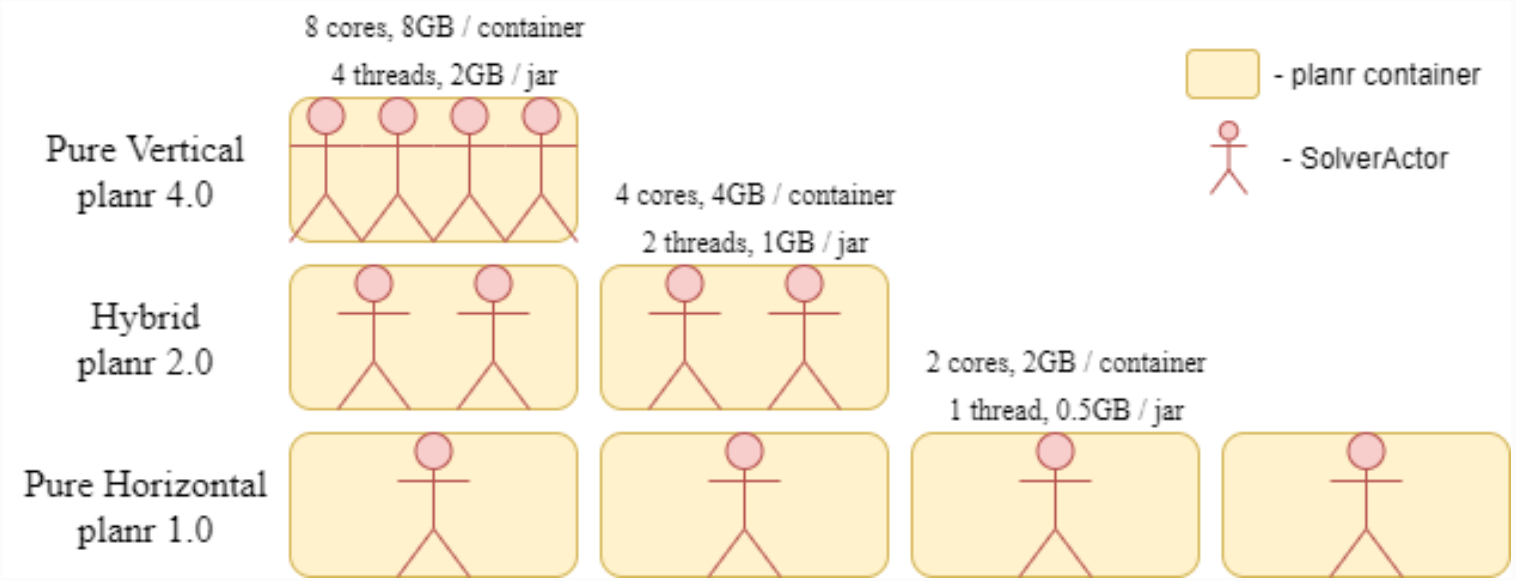

Gatling
Docker
Horizontal scaling
nginx
planr
SolverActor
planr
SolverActor
planr
SolverActor
planr
SolverActor
Gatling
Docker
Vertical scaling
nginx
planr
SolverActor

Initial scaling scenarios

/	C	Co/C	M/C (GB)	A/C	H/C (GB)
planr 4.0	1	8	8	4	2
planr 2.0	2	4	4	2	1
planr 1.0	4	2	2	1	0.5

(C – Container, Co – Core, M – Memory, A – Akka Actor/Thead, H – JVM Heap)

Initial scaling scenarios

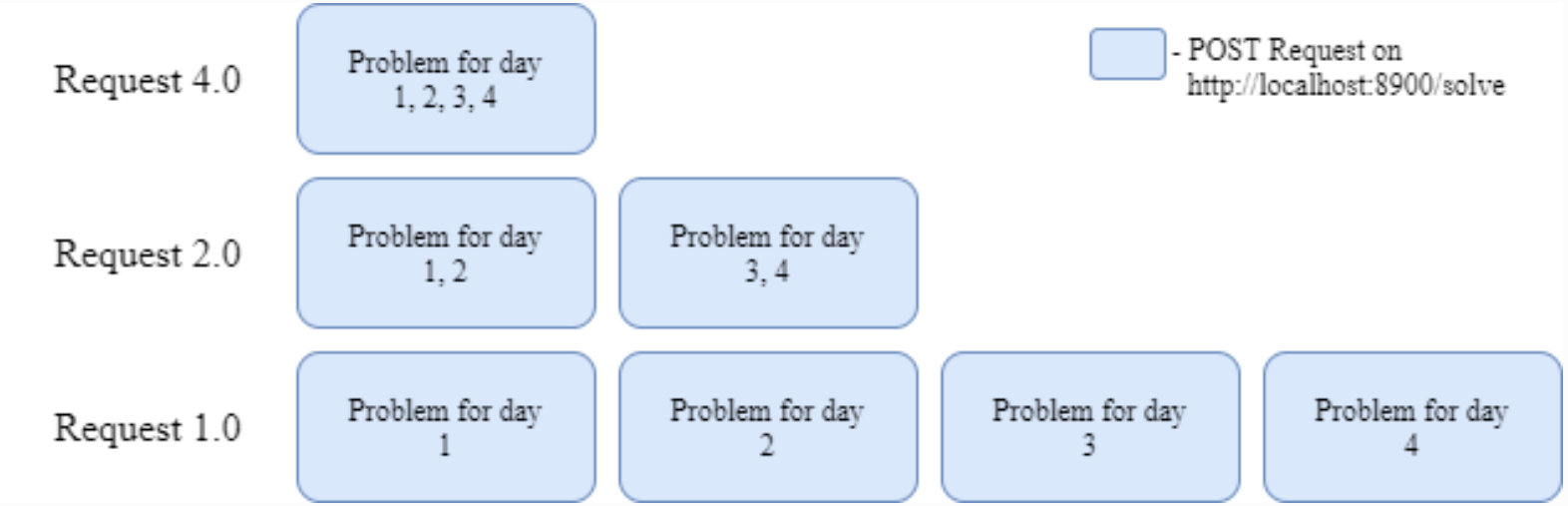


Request scenarios

/	R	P/R	S/R
Request 4.0	1	4	4
Request 2.0	2	2	2
Request 1.0	4	1	1

(R – POST request, P – Problem, S – Solution)

Request scenarios

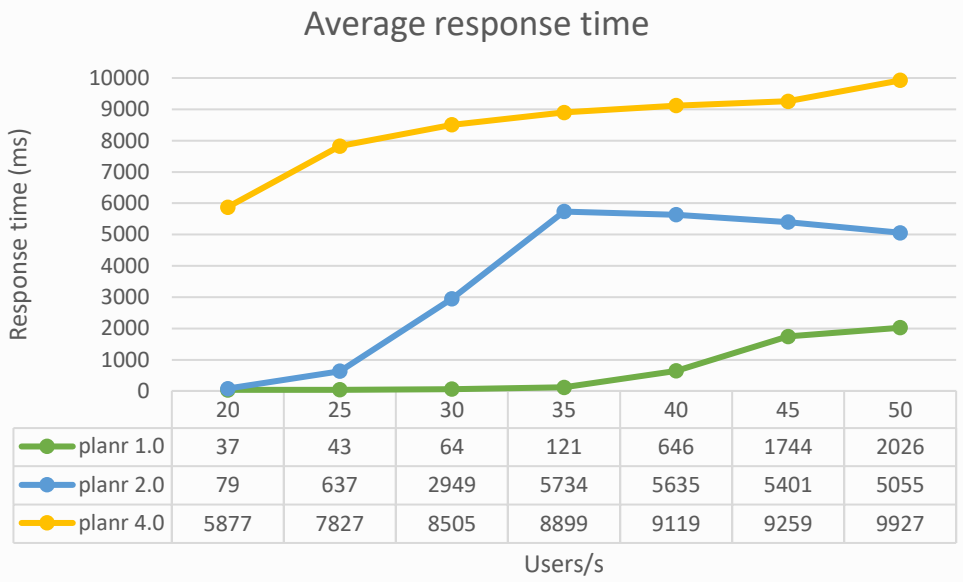


Initial test scenarios

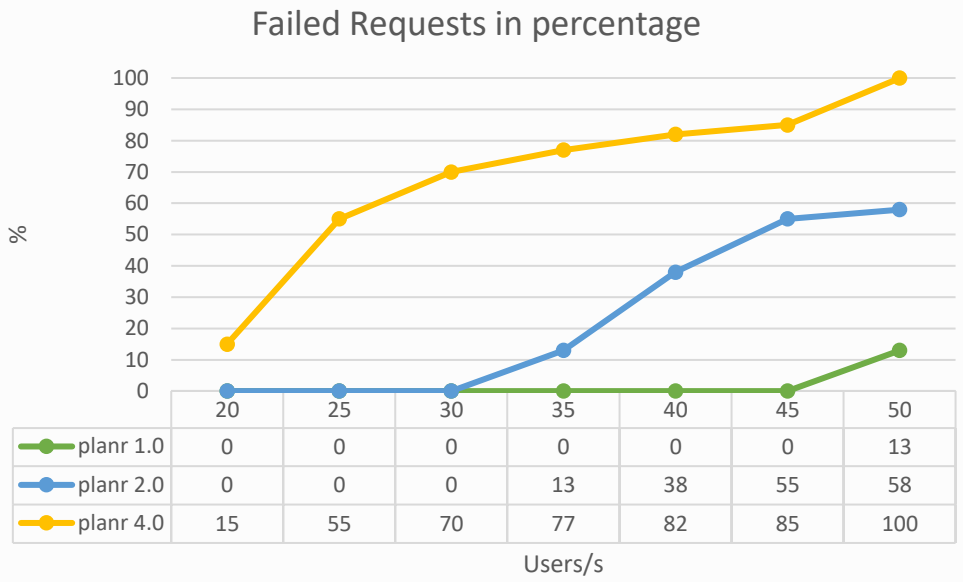
/	planr 1.0	planr 2.0	planr 4.0
Scenario 1	Request 1.0	Request 2.0	Request 4.0
Scenario 2	Request 1.0	Request 1.0	Request 1.0
Scenario 3	Request 4.0	Request 4.0	Request 4.0

Results

Scenario 1 (c)



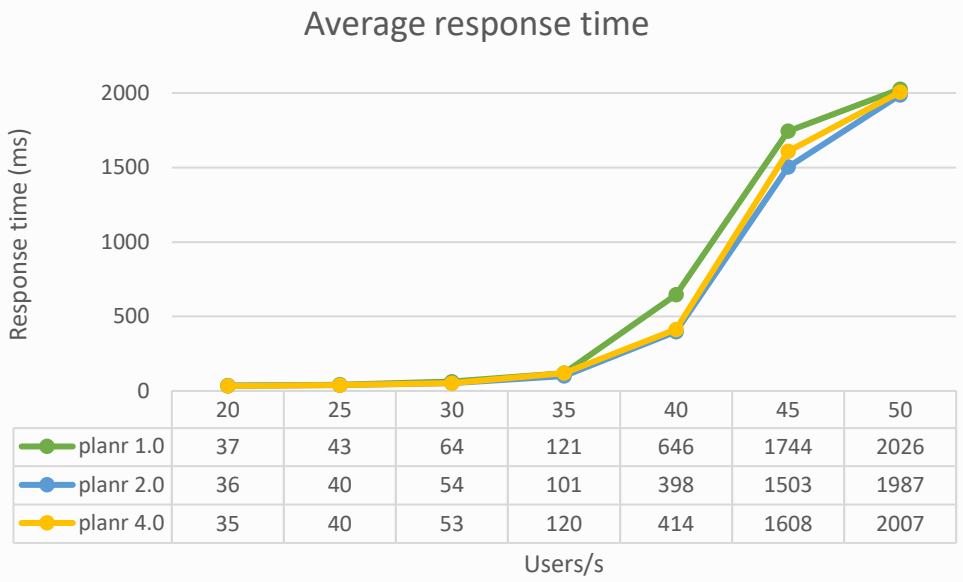
	20	25	30	35	40	45	50
planr 1.0	37	43	64	121	646	1744	2026
planr 2.0	79	637	2949	5734	5635	5401	5055
planr 4.0	5877	7827	8505	8899	9119	9259	9927



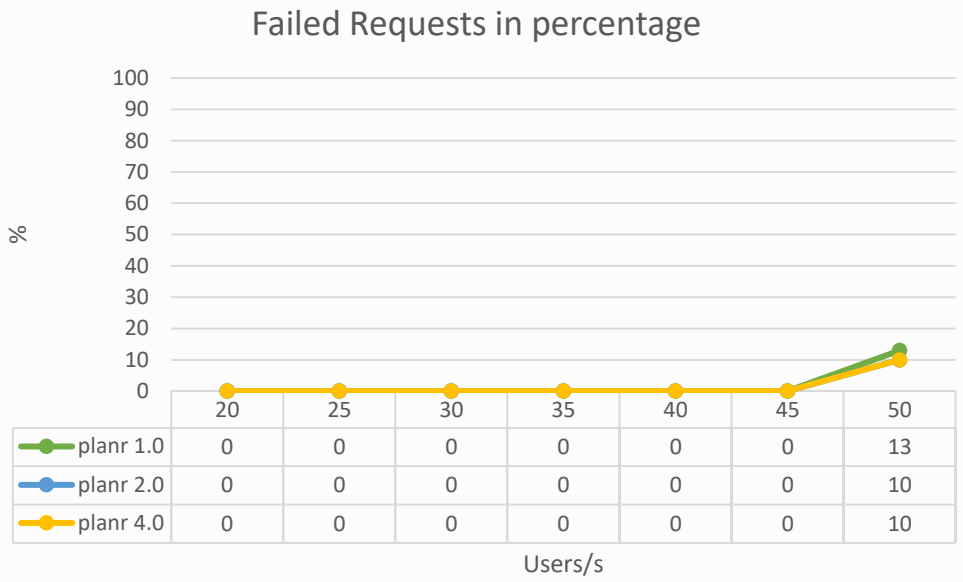
	20	25	30	35	40	45	50
planr 1.0	0	0	0	0	0	0	13
planr 2.0	0	0	0	13	38	55	58
planr 4.0	15	55	70	77	82	85	100

Results

Scenario 2 (h)



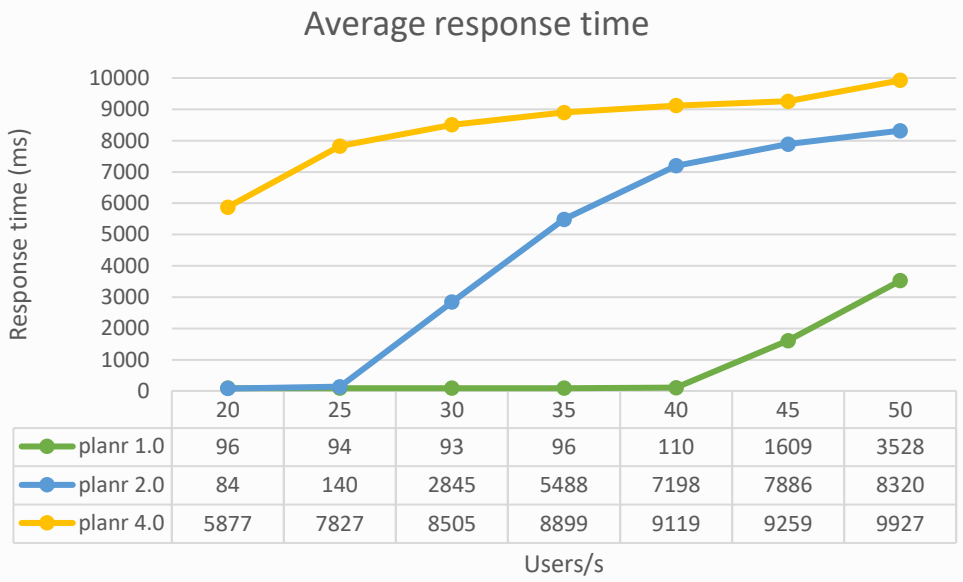
	20	25	30	35	40	45	50
planr 1.0	37	43	64	121	646	1744	2026
planr 2.0	36	40	54	101	398	1503	1987
planr 4.0	35	40	53	120	414	1608	2007



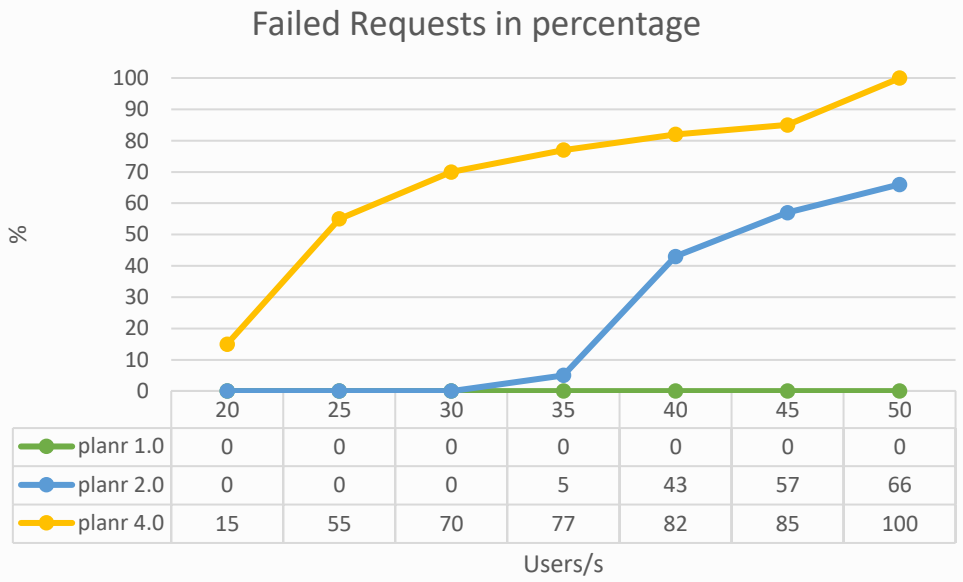
	20	25	30	35	40	45	50
planr 1.0	0	0	0	0	0	0	13
planr 2.0	0	0	0	0	0	0	10
planr 4.0	0	0	0	0	0	0	10

Results

Scenario 3 (v)



	20	25	30	35	40	45	50
planr 1.0	96	94	93	96	110	1609	3528
planr 2.0	84	140	2845	5488	7198	7886	8320
planr 4.0	5877	7827	8505	8899	9119	9259	9927



	20	25	30	35	40	45	50
planr 1.0	0	0	0	0	0	0	0
planr 2.0	0	0	0	5	43	57	66
planr 4.0	15	55	70	77	82	85	100

Further scaling scenarios

/	C	Co/C	M/C (GB)	A/C	H/C (GB)
planr 1.2	4	2	2	2	0.5
planr 1.4	4	2	2	4	0.5
planr 1.8	4	2	2	8	0.5
planr 1.16	4	2	2	16	0.5

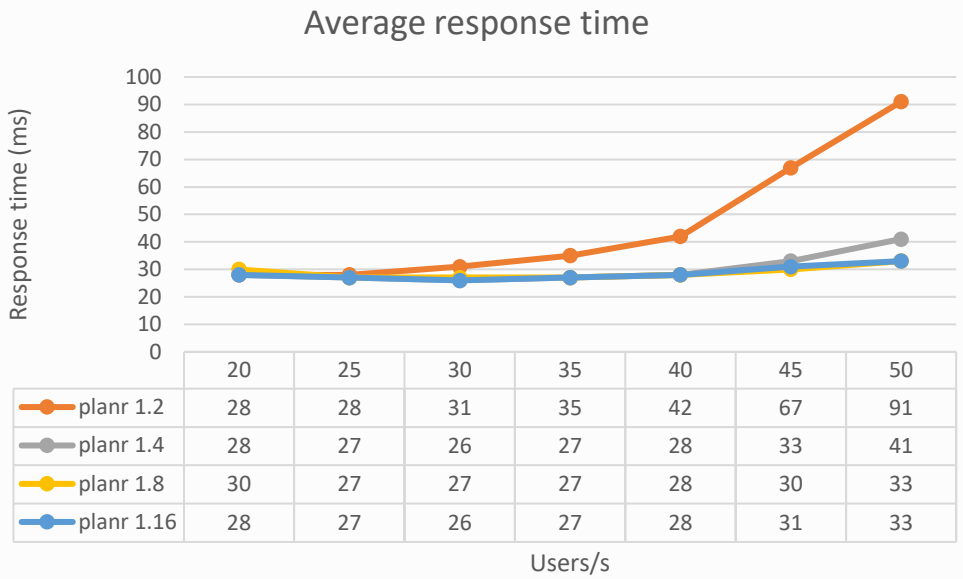
(C – Container, Co – Core, M – Memory, A – Akka Actor/Thead, H – JVM Heap)

Further test scenarios

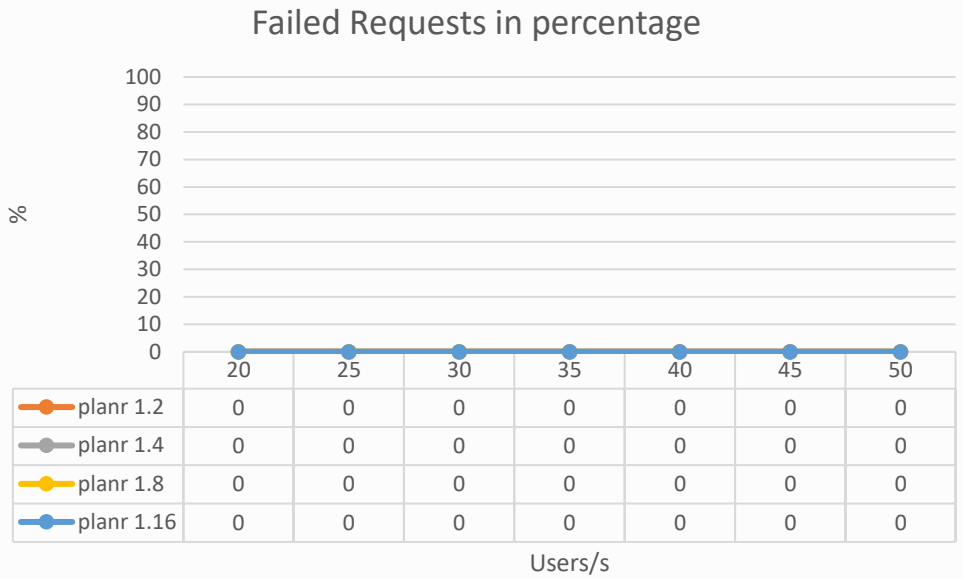
/	planr 1.2	planr 1.4	planr 1.8	planr 1.16
Scenario 4	Request 1.0	Request 1.0	Request 1.0	Request 1.0
Scenario 5	Request 4.0	Request 4.0	Request 4.0	Request 4.0

Results

Scenario 4 (h)



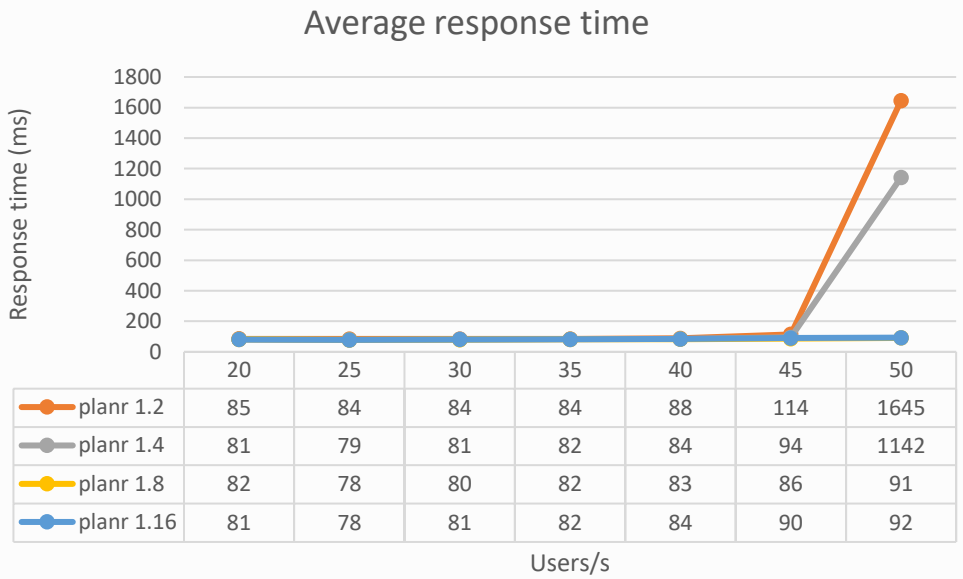
	20	25	30	35	40	45	50
planr 1.2	28	28	31	35	42	67	91
planr 1.4	28	27	26	27	28	33	41
planr 1.8	30	27	27	27	28	30	33
planr 1.16	28	27	26	27	28	31	33



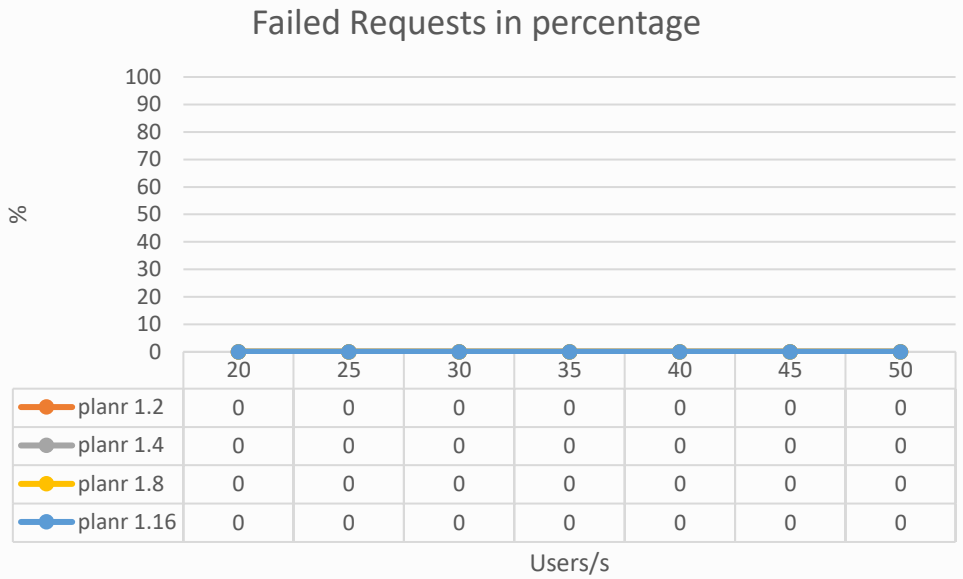
	20	25	30	35	40	45	50
planr 1.2	0	0	0	0	0	0	0
planr 1.4	0	0	0	0	0	0	0
planr 1.8	0	0	0	0	0	0	0
planr 1.16	0	0	0	0	0	0	0

Results

Scenario 5 (v)



	20	25	30	35	40	45	50
planr 1.2	85	84	84	84	88	114	1645
planr 1.4	81	79	81	82	84	94	1142
planr 1.8	82	78	80	82	83	86	91
planr 1.16	81	78	81	82	84	90	92



	20	25	30	35	40	45	50
planr 1.2	0	0	0	0	0	0	0
planr 1.4	0	0	0	0	0	0	0
planr 1.8	0	0	0	0	0	0	0
planr 1.16	0	0	0	0	0	0	0

Conclusions

- Pure horizontal scaling provides better performance than a pure vertical one
- The combination of scale-out and scale-up techniques maximizes efficiency to cope with overload periods
- Future work consists of defining the relation between the vertical scaling entities and physical resources

Thank you for your attention!