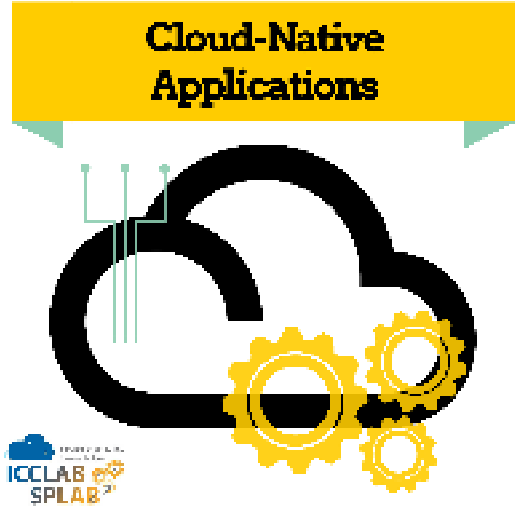
Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices

Manuel Ramírez López <ramz@zhaw.ch> &
Josef Spillner <josef.spillner@zhaw.ch>
Service Prototyping Lab (blog.zhaw.ch/icclab)

Dec 5, 2017 | 6th CloudAM @ UCC, Austin, TX, USA

Motivation



Application...

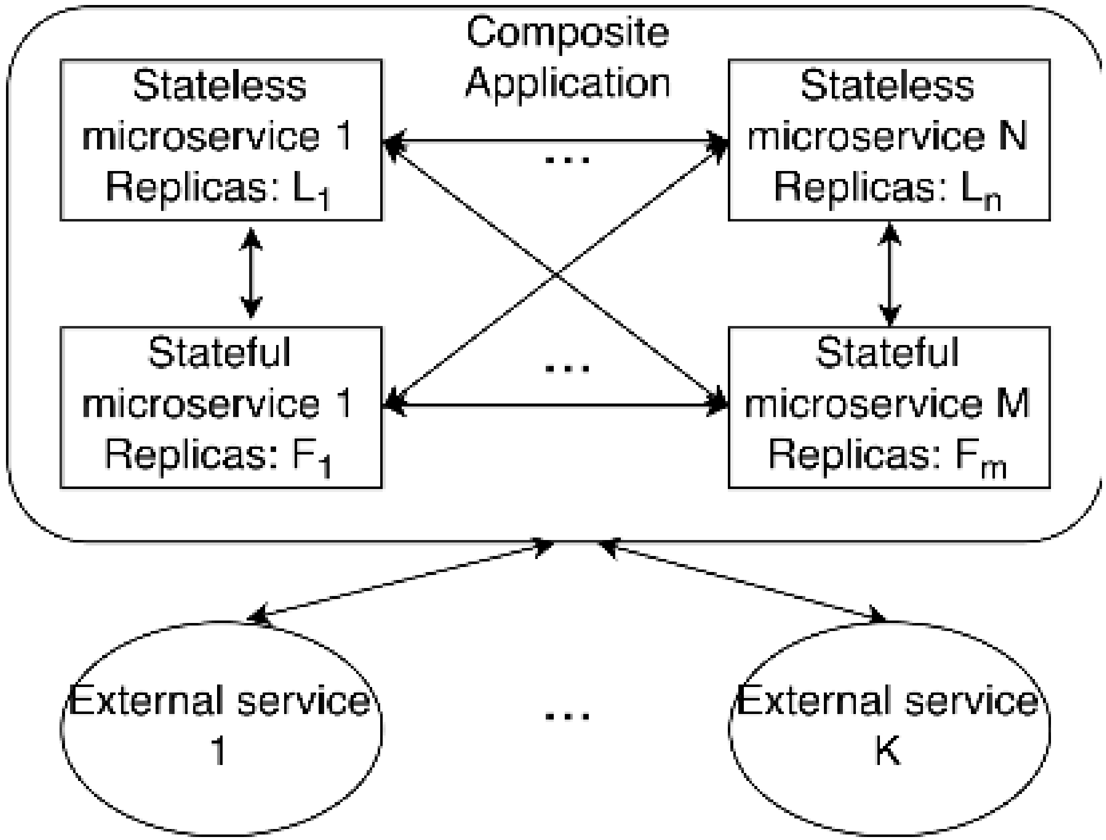
... scaling:
- accomodate more users / growing workload
- desired: elasticity, rapidity

... auto-scaling:
- rule-based scaling actions
- ↹ trade-offs: effort for rule definition, initial calibration, hotspots

... pre-scaling (our work):
- determine initial combinatorial scaling
- fixed-workload vs. variable workload

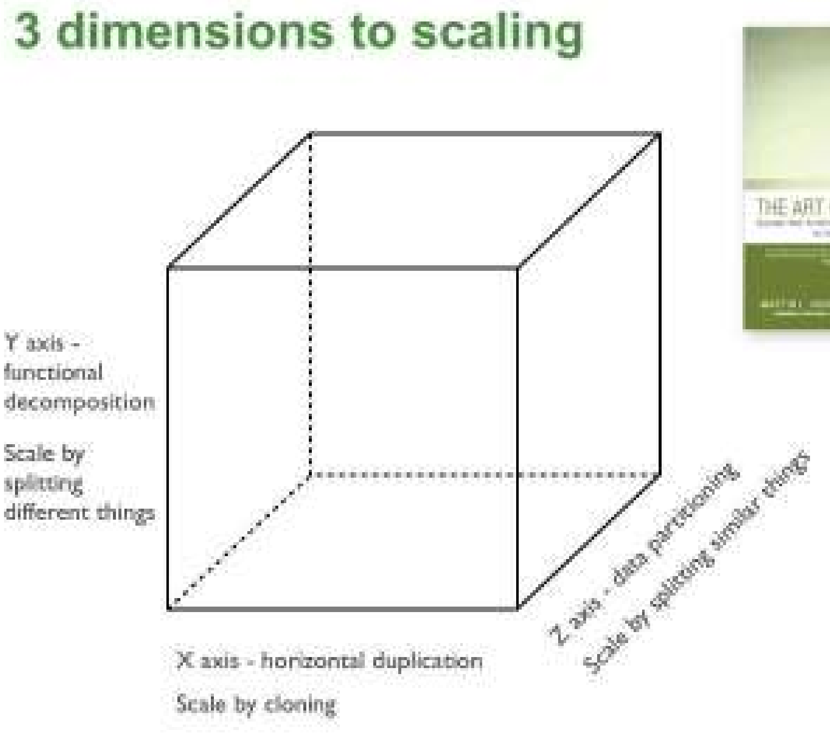
Model

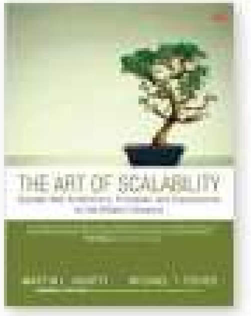
Microservices composition - three classes of services



Model

Scale cube (Abbott and Fisher, 2015)





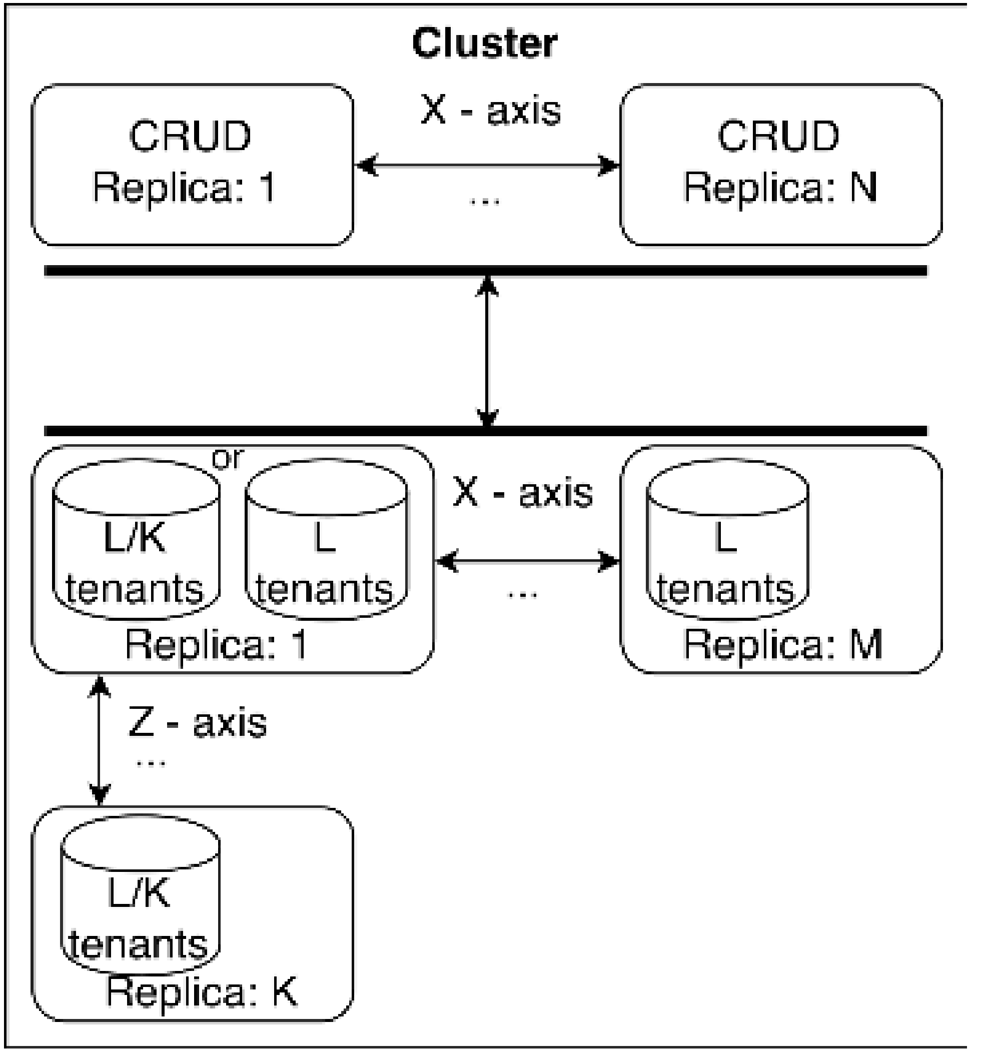
Independently deployable microservices → Y axis (Hasselbring, 2016)

Assumptions

Application architecture following a microservice design
- stateful CRUD service
- replica count per service

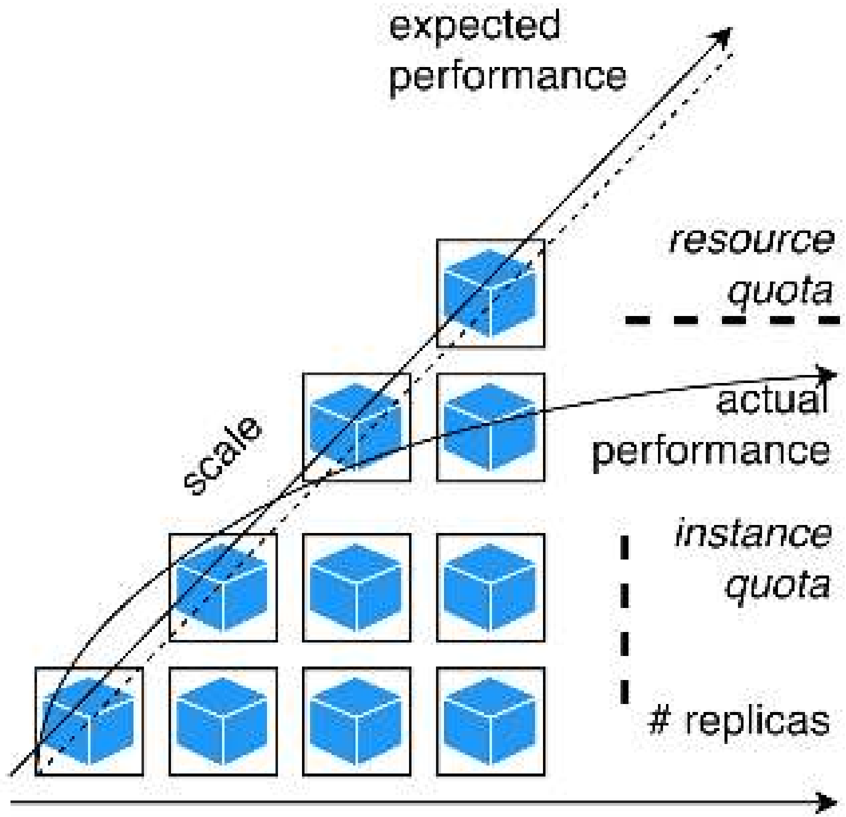
Scenario implementation
- online document management application
- RESTful Python service, MongoDB

Scale cube relation
- X axis: horizontal replication
- Z axis: data partitions



Assumptions

Nonlinear constrained horizontal scaling behaviour on X axis according to following graph



Research Question and Approach

Question:

»Can the best combination of replicas for a given application and workload be calculated for performance-critical and cost-constrained settings?«

application X workload X optimality

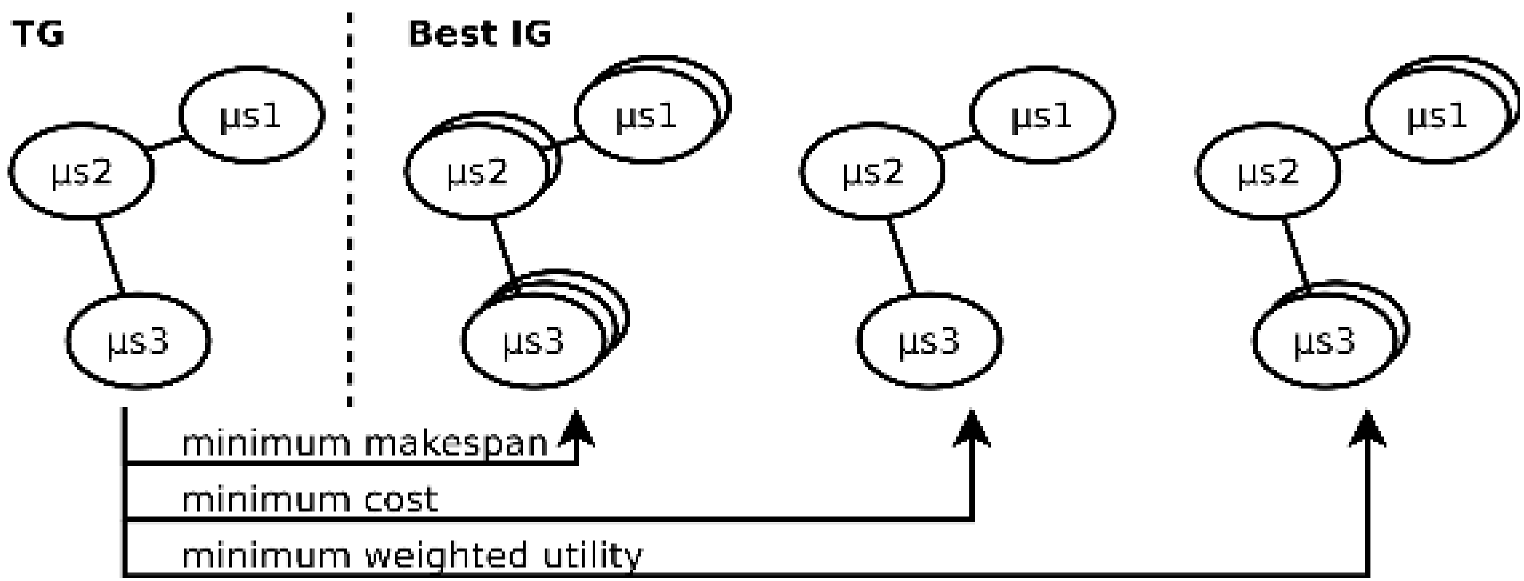
Approach:

- Formalisation of application structure, task, workload, environment + scaling constraints
- Combinations of scaling factors, optimal result vectors

Method: Optimality

What is the “best“ combination?

TG: Type graph
IG: Instance graph - replicas per microservice - 3x IG_O



Method: Formalisation

Mathematical model: m-dimensional makespan matrix

$$M_e = M(e)_{n_1 \times \cdots \times n_m} = \begin{pmatrix} \mu_{1,1,\ldots} & \mu_{1,2,\ldots} & \cdots & \mu_{1,n_2,\ldots} \\ \mu_{2,1,\ldots} & \mu_{2,2,\ldots} & \cdots & \mu_{2,n_2,\ldots} \\ \cdots & \cdots & \cdots & \cdots \\ \mu_{n_1,1,\ldots} & \mu_{n_1,2,\ldots} & \cdots & \mu_{n_1,n_2,\ldots} \end{pmatrix}$$

(2 out of m dimensions shown conforming scenario)

where:

- m - # of microservices
- n - # of replicas per microservice
  - stateful services: partitioning scheme (e.g. per tenant)
- e - experiment (task/workload combination)
- µ - makespan

ICCLAB Service Engineering Research Area SPLAB

Method: Optimal Factors Formula

Three approaches

- unconstrained (baseline)
- constrained
- relaxed-constrained (with rate)

$$fastest(M_e, prices, max_\mu, max_\kappa) = i \mid \min_{\forall i \in I}\{m_i \in M_e \mid m_i < max_\mu, cost(i, prices) < max_\kappa\}$$

$$cheapest(M_e, prices, max_\mu, max_\kappa) = i \mid \min_{\forall i \in I}\{cost(i, prices) \mid M_e \ni m_i < max_\mu, cost(i, prices) < max_\kappa\}$$

cost: resource cost or monetary cost

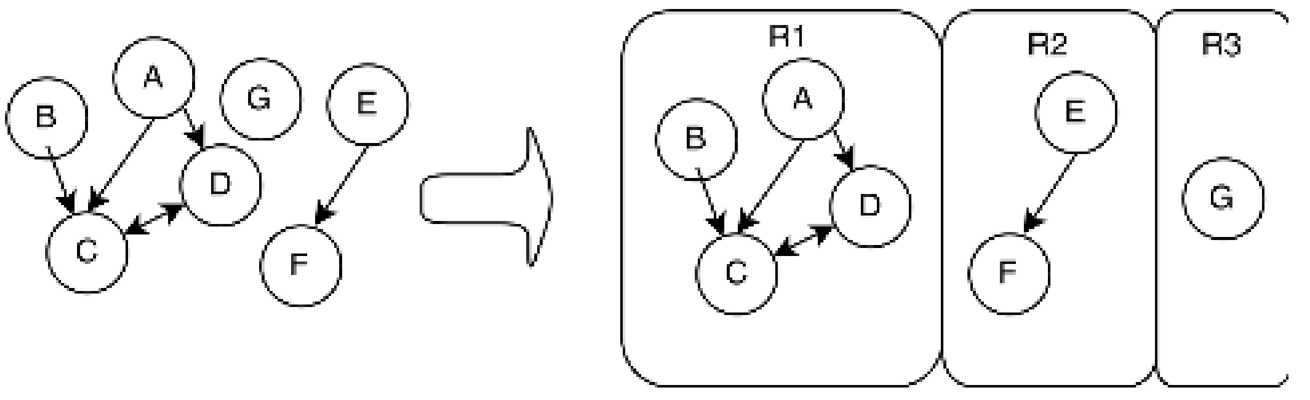
I: set of indices of M

Method: Complexity Reduction

Sparse matrices/arrays due to not fully connected microservices (TG level)

- representation: bi-directional disconnected graph
- vertices = microservices
- edges = connections (communication links)

Transformation: set of fully connected graphs



(caveat: not validated, relates to patterns - e.g. sidecar)

Implementation: Factor Injection

Integration with microservice management platforms

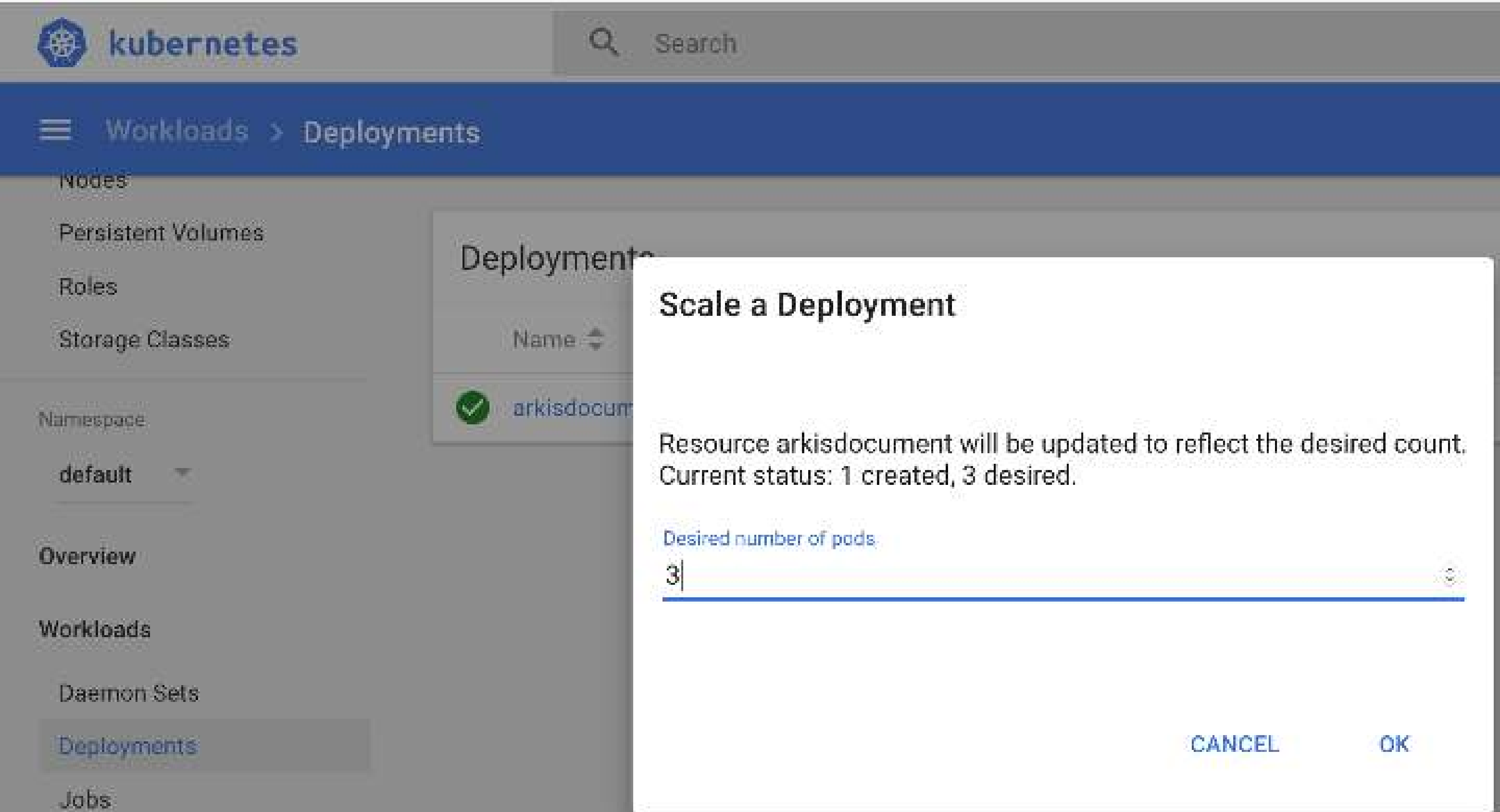
- e.g. container schedulers (Docker Compose, Kubernetes, ...)
- using placeholders in composition templates

Example as used in experiment:
Kubernetes 1.5 deployment @ Google Cloud Platform (GCP - GCE)

```
{
 "kind": "Deployment",
 "apiVersion": "extensions/v1beta1",
 "metadata": {
  "name": "MICROSERVICE",
 },
 "spec": {
  "replicas": REPLICAS,
   "spec": {
    "containers": [
     {
      "name": "MICROSERVICEIMPL",
      "image": "NAMESPACE/CONTAINER:1.1",
...
```

Implementation: Factor Injection

Verification through graphical user interface



Results

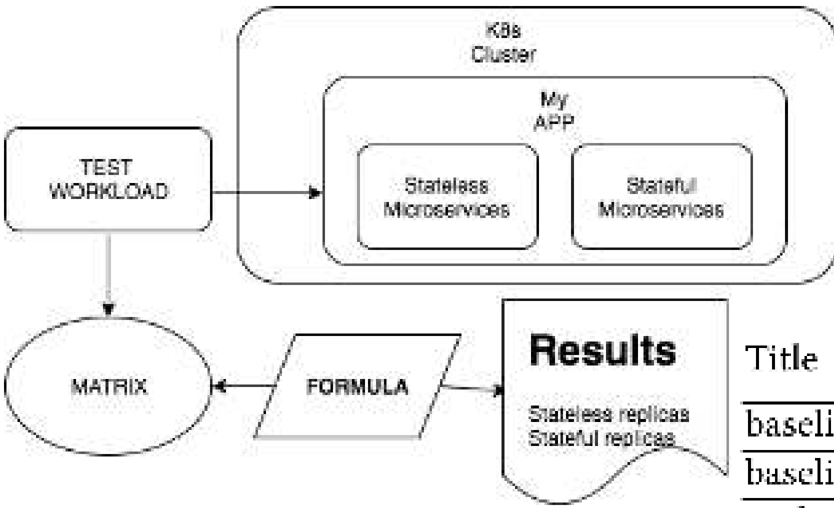
Stateless microservice: “arkisdocument“, API to search in documents

- from 1 to 11 replicas

Stateful microservice “mongodb“, 300 documents per tenant

- from 1 to 2 replicas

Workload generator/test microservice, not managed, not scaled



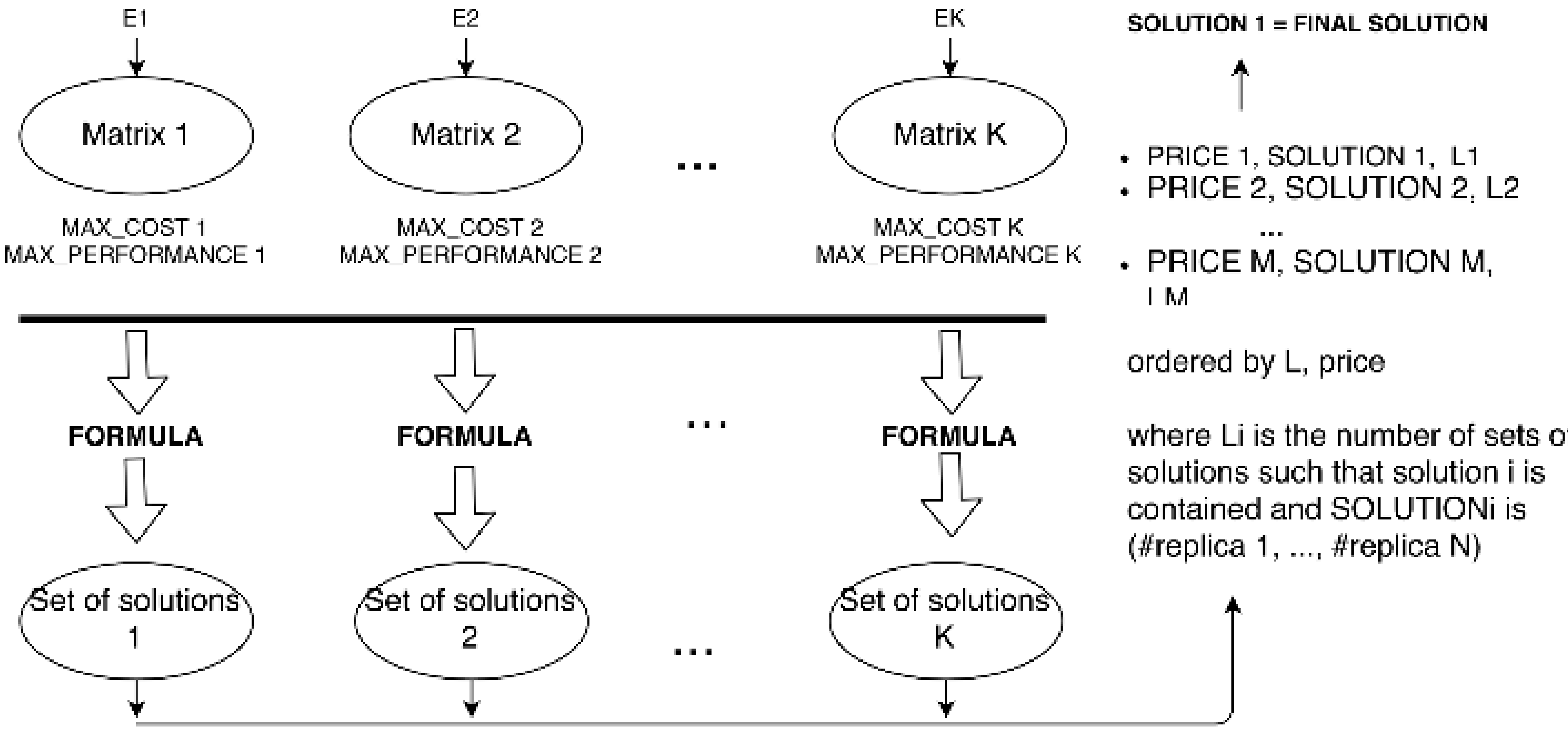
Title	Policy	max_{μ}	max_{κ}	Rate	#S-ful	#S-less	Cost	Makespan
baseline	fastest	X	X	X	2	7	0.83	35.92
baseline	cheapest	X	X	X	1	1	0.33	89.16
with C	fastest	45.0	0.8	X	2	5	0.75	40.07
with C	cheapest	45.0	0.8	X	1	5	0.5	43.79
with C&R	fastest	45.0	0.8	1.06	1	7	0.58	41.88
with C&R	cheapest	45.0	0.8	1.2	1	7	0.58	41.88

Legend: S-less = stateless, S-ful = stateful, C = constraints, R = rate

Cost/performance ratio is not linear.

Outlook: Variable Workloads

K experiments with maximum fulfilment of cost/performance requirements
Intersection analysis



Summary

Contributions

- formalised application scaling determination (X + Z axes in scale cube with microservice composition as Y axis)
- testbed based on Docker containers in Kubernetes
- practical use to complement autoscaling
- scientific open notebook for future work

`https://github.com/serviceprototypinglab/scalability-experiments`

Recent related work: «ThrottleBot - Performance without Insight» by Chang, Panda, Tsai, Wang, Shenker (arXiv:1711.00618)

«Microservices for Scalability» by Wilhelm Hasselbring, ICPE'16 keynote