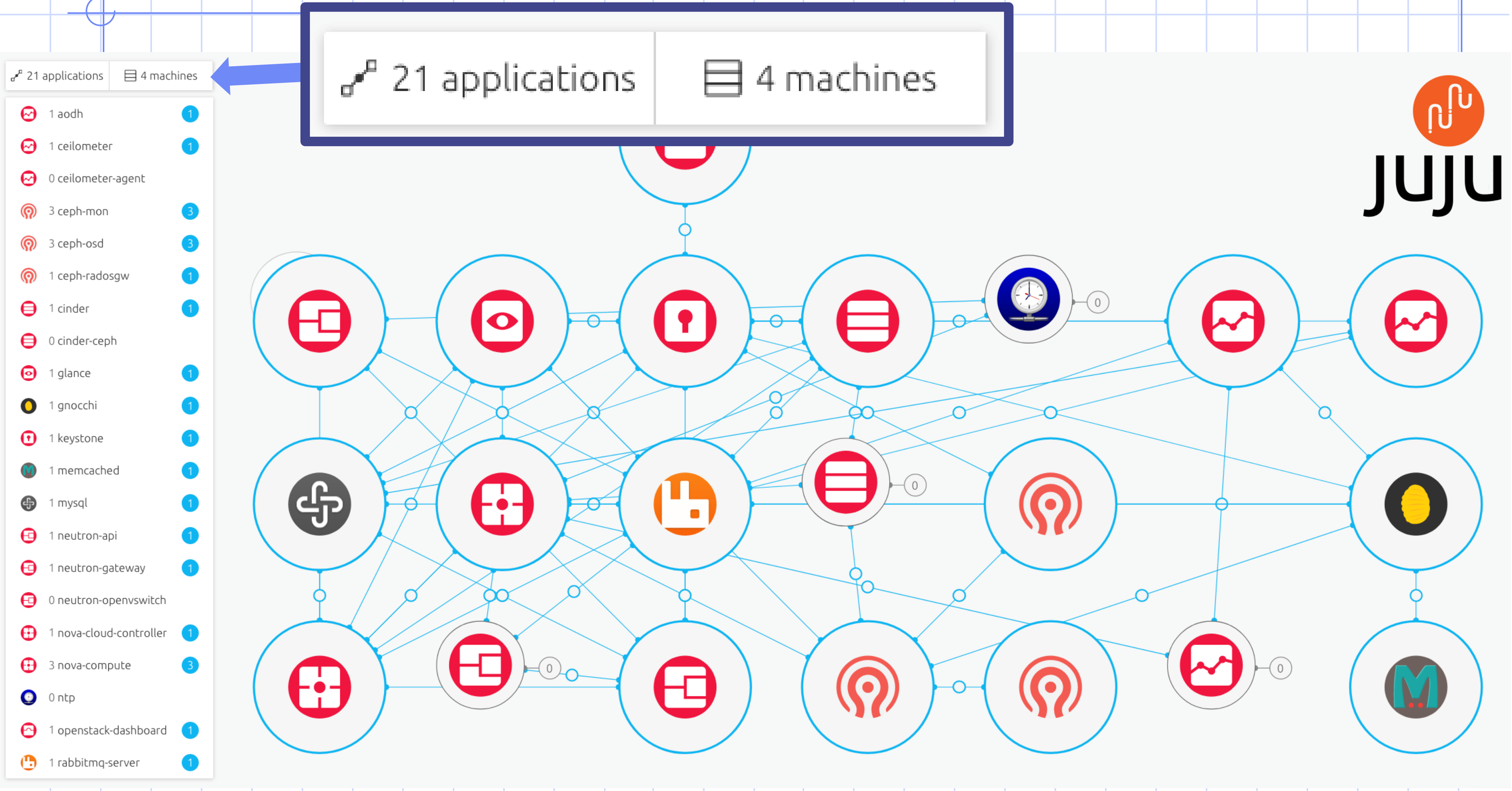
Optimal and Automated Deployment for Microservices

Lorenzo Bacchiani Univ. of Bologna - Italy

Joint work with:
M. Bravetti, S. Giallorenzo, I. Talevi and G. Zavattaro
Univ. of Bologna - Italy

J. Mauro Southern Denmark Univ. - Denmark

Deploying component-based applications is complex



Automatic synthesis of deployment orchestration

- Mastering such complexity requires **automation**
- Deployment orchestration synthesis requires specific knowledge:
  - Component **dependencies**
  - Component **configuration life-cycle**

(similar to dependency-conflicts metadata used in the automatic configuration of **package-based software distributions**)

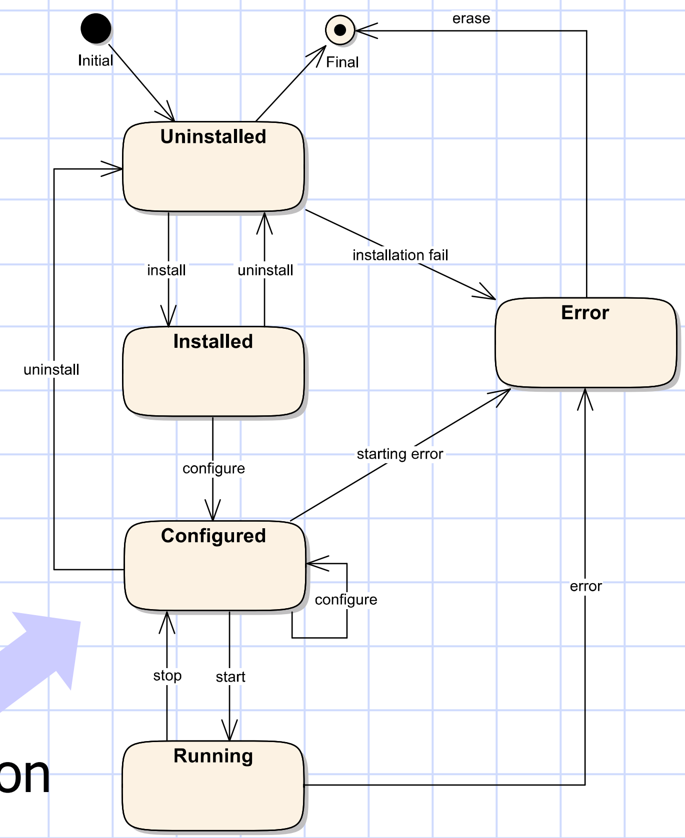
Dependencies and configuration life-cycle

CloudMF

[N.Ferry et al. – ACM ToIT 18]

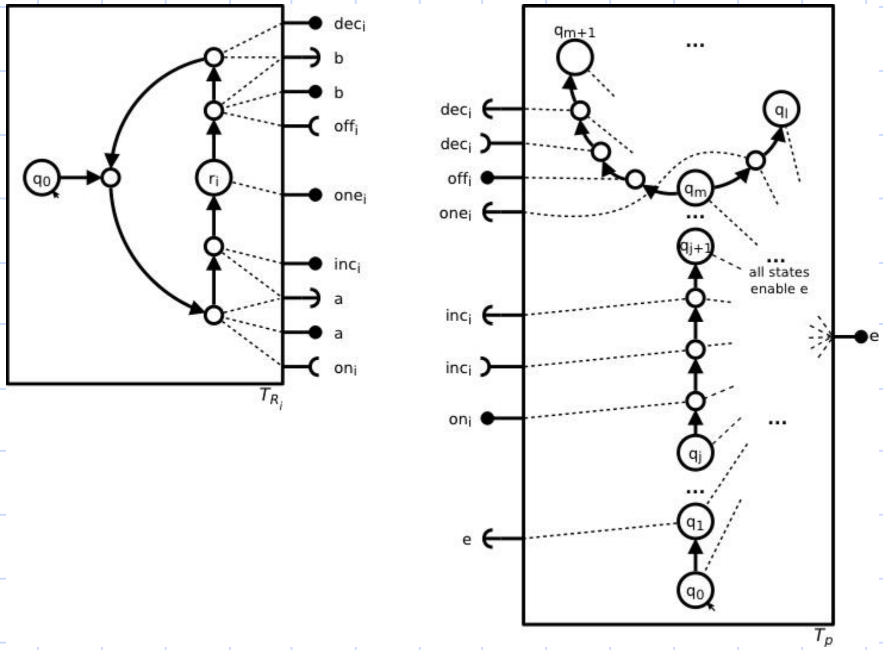
```
{
  "id" : "SensApp",
  "retrieval" : "wget -P ~ http://cloudml.org/SensApp.
       war; wget -P ~ http://cloudml.org/startsensapp.
       sh ; wget -P ~ http://cloudml.org/deploysensapp.
       sh",
  "deployment" : "deploysensapp.sh",
  "start" : "startsensapp.sh",
  "requires" : [
    { "id" : "JettyCapability", "isOptional" : false },
    { "id" : "MongoDBCapability", "isOptional" : false
         }
  ],
  "inputs" : [
    {
      "id" : "RESTChannel",
      "portNumber" : "8080",
      "isRemote" : true
    }
  ],
  "provides" : [
    {
      "id" : "RESTServer",
      "portNumber" : "8080"
    }
  ]
}
```

Dependencies



Configuration life-cycle

Is the automated deployment problem decidable?



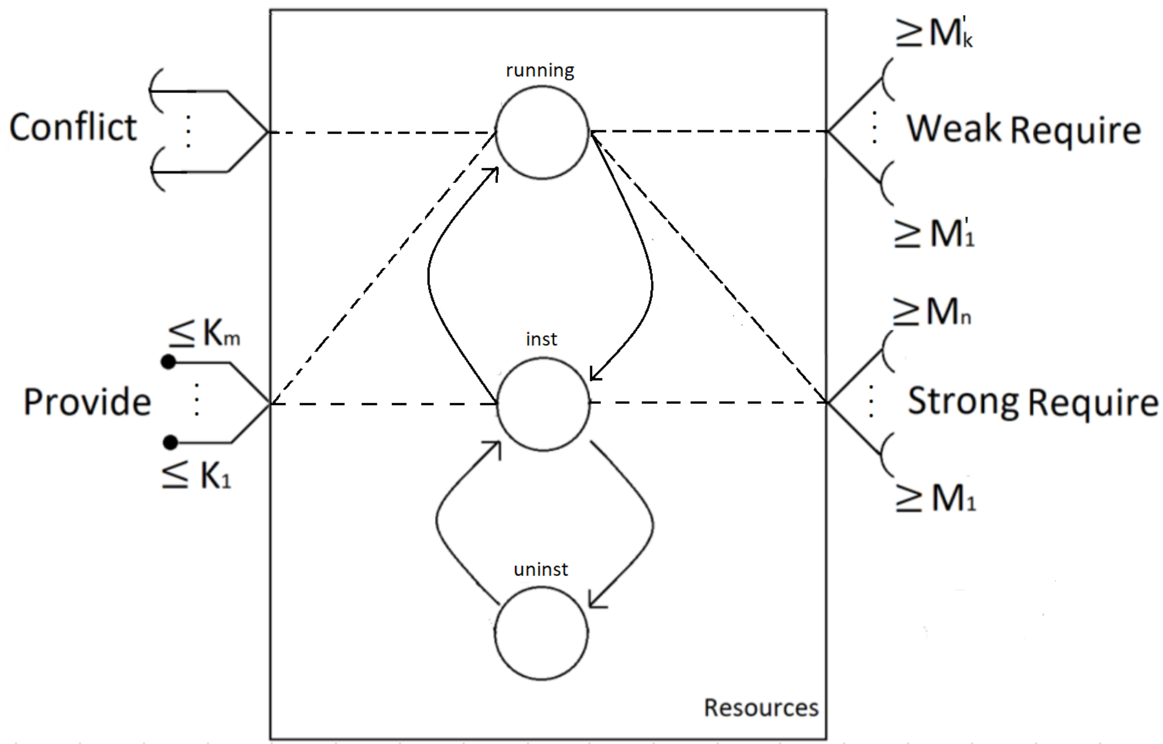
- We saw that such complexity causes orchestration synthesis to be uncedidable

What if components are Microservices?

- Microservices
  - Components become:
    - **Fine grained**
    - **Loosely coupled**
  - This facilitates:
    - Development (simple components)
    - Maintenance (local modifications)
    - …
- What about **deployment**?

Microservice's finite state automaton

- **Fixed state machine** states/transitions



State-of-the-art microservice deployment technologies

```
version: '3'
services:
 web:
  build: .
  image: some-image
  ports:
         - "3001:3000"
  dns: "8.8.8.8"
  volumes:
   - ".:/app"
  env_file: .env
     links:
  - redis:redis
     external_links:
   - postgres1
  ...
```

Two types of dependencies:

- links

force an order of activation

- external links

order of activation does not matter

State-of-the-art microservice deployment technologies

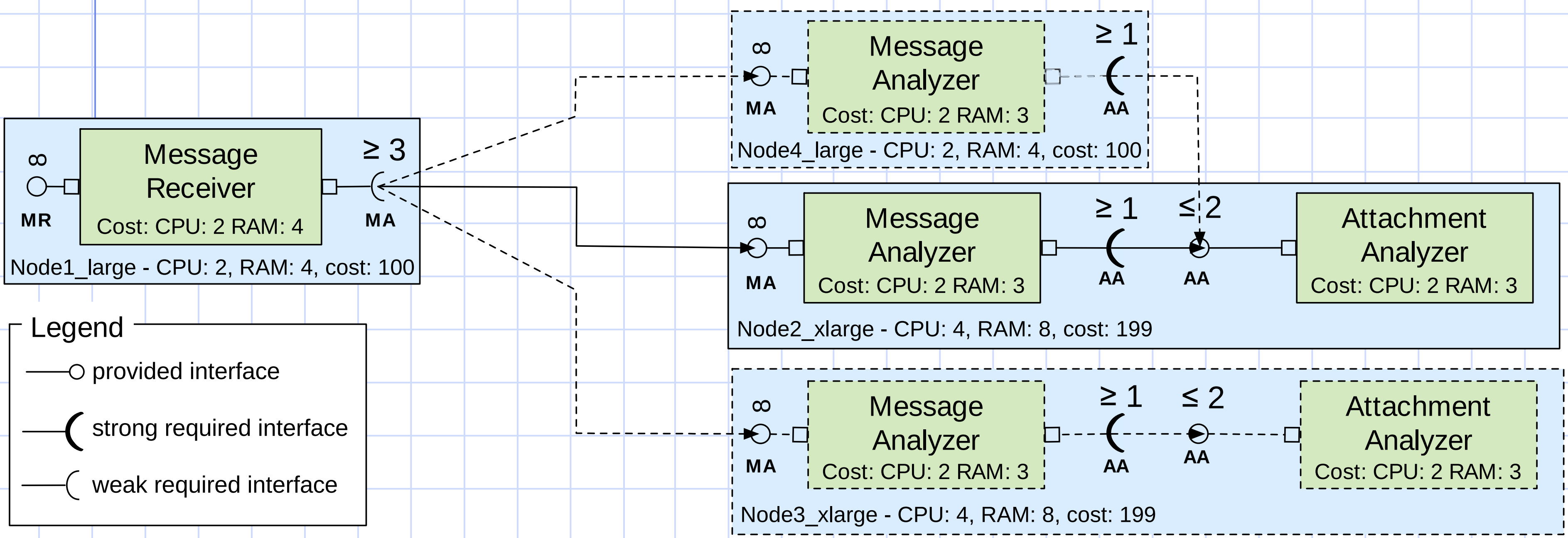
```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
  ...
```

kubernetes

Services (pods) consume resources

Hosting nodes must provide pods with such resources

A simplified model for microservice deployment



Main result

- Deployment orchestration synthesis is **decidable**
  - Proof:
    - translation of the problem in **sets of constraints** to be given in input to a constraint solver/optimizer
  - Side effect:
    - optimization functions (e.g. minimize total cost) can be used to **optimize** some metrics

The algorithm

- Step 1:
  compute service **instances** and their **distribution** over computing nodes

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U,\ p\in\mathtt{dom}(\mathcal{T}.\mathtt{reqs})} \mathcal{T}.\mathtt{reqs}(p)\cdot\mathtt{inst}(\mathcal{T}) \le \sum_{\mathcal{T}'\in U}\mathtt{bind}(p,\mathcal{T},\mathcal{T}')$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U,\ p\in\mathtt{dom}(\mathcal{T}.\mathtt{reqw})} \mathcal{T}.\mathtt{reqw}(p)\cdot\mathtt{inst}(\mathcal{T}) \le \sum_{\mathcal{T}'\in U}\mathtt{bind}(p,\mathcal{T},\mathcal{T}')$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U,\ \mathcal{T}.\mathtt{prov}(p)<\infty} \mathcal{T}.\mathtt{prov}(p)\cdot\mathtt{inst}(\mathcal{T}) \ge \sum_{\mathcal{T}'\in U}\mathtt{bind}(p,\mathcal{T}',\mathcal{T})$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U,\ \mathcal{T}.\mathtt{prov}(p)=\infty} \mathtt{inst}(\mathcal{T})=0 \ \Rightarrow\ \sum_{\mathcal{T}'\in U}\mathtt{bind}(p,\mathcal{T}',\mathcal{T})=0$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U,\ p\notin\mathtt{dom}(\mathcal{T}.\mathtt{prov})} \sum_{\mathcal{T}'\in U}\mathtt{bind}(p,\mathcal{T}',\mathcal{T})=0$$

$$\mathtt{inst}(\mathcal{T}_t)\ge 1$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\substack{\mathcal{T}\in U,\\ p\in\mathcal{T}.\mathtt{conf}}} \bigwedge_{\substack{\mathcal{T}'\in U-\{\mathcal{T}\},\\ p\in\mathtt{dom}(\mathcal{T}'.\mathtt{prov})}} \mathtt{inst}(\mathcal{T})>0 \ \Rightarrow\ \mathtt{inst}(\mathcal{T}')=0$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\substack{\mathcal{T}\in U,\ p\in\mathcal{T}.\mathtt{conf}\ \wedge\\ p\in\mathtt{dom}(\mathcal{T}.\mathtt{prov})}} \mathtt{inst}(\mathcal{T})\le 1$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U} \bigwedge_{\mathcal{T}'\in U-\{\mathcal{T}\}} \mathtt{bind}(p,\mathcal{T},\mathcal{T}')\le\mathtt{inst}(\mathcal{T})\cdot\mathtt{inst}(\mathcal{T}')$$

$$\bigwedge_{p\in\mathcal{I}(U)} \bigwedge_{\mathcal{T}\in U} \mathtt{bind}(p,\mathcal{T},\mathcal{T})\le\mathtt{inst}(\mathcal{T})\cdot(\mathtt{inst}(\mathcal{T})-1)$$

NP-complete

$$\mathtt{inst}(\mathcal{T})=\sum_{o\in O}\mathtt{inst}(\mathcal{T},o)$$

$$\bigwedge_{r\in\mathcal{R}}\bigwedge_{o\in O}\sum_{\mathcal{T}\in U}\mathtt{inst}(\mathcal{T},o)\cdot\mathcal{T}.\mathtt{res}(r)\le o.\mathtt{res}(r)$$

$$\bigwedge_{o\in O}\Big(\sum_{\mathcal{T}\in U}\mathtt{inst}(\mathcal{T},o)>0\Big)\Leftrightarrow\mathtt{used}(o)$$

$$\min\sum_{o\in O,\ \mathtt{used}(o)} o.\mathtt{cost}$$

The algorithm

- Step 2:
  defining **connections** among instances

$$\bigwedge_{\mathcal{T}\in U}\ \bigwedge_{p\in\mathcal{I}(U)}\ \bigwedge_{i\in 1\dots n}\ \sum_{j\in(1\dots m)\setminus\{i|\mathcal{T}=\mathcal{T}'\}} \mathtt{b}(p, s_i^{\mathcal{T}}, s_j^{\mathcal{T}'}) \leq limProv(\mathcal{T}', p)$$

$$\bigwedge_{\mathcal{T}\in U}\ \bigwedge_{p\in\mathtt{dom}(\mathcal{T}.\mathtt{req_S})}\ \bigwedge_{i\in 1\dots n}\ \sum_{j\in(1\dots m)\setminus\{i|\mathcal{T}=\mathcal{T}'\}} \mathtt{b}(p, s_i^{\mathcal{T}}, s_j^{\mathcal{T}'}) \geq \mathcal{T}.\mathtt{req_S}(p)$$

$$\bigwedge_{\mathcal{T}\in U}\ \bigwedge_{p\in\mathtt{dom}(\mathcal{T}.\mathtt{req_W})}\ \bigwedge_{i\in 1\dots n}\ \sum_{j\in(1\dots m)\setminus\{i|\mathcal{T}=\mathcal{T}'\}} \mathtt{b}(p, s_i^{\mathcal{T}}, s_j^{\mathcal{T}'}) \geq \mathcal{T}.\mathtt{req_W}(p)$$

$$\bigwedge_{\mathcal{T}\in U}\ \bigwedge_{p\notin\mathtt{dom}(\mathcal{T}.\mathtt{req_S})\cup\mathtt{dom}(\mathcal{T}.\mathtt{req_W})}\ \bigwedge_{i\in 1\dots n}\ \sum_{j\in(1\dots m)\setminus\{i|\mathcal{T}=\mathcal{T}'\}} \mathtt{b}(p, s_i^{\mathcal{T}}, s_j^{\mathcal{T}'}) = 0$$

Step 3:
orchestration synthesis (topological sort, assuming no circular strong dependencies)

EXPTIME

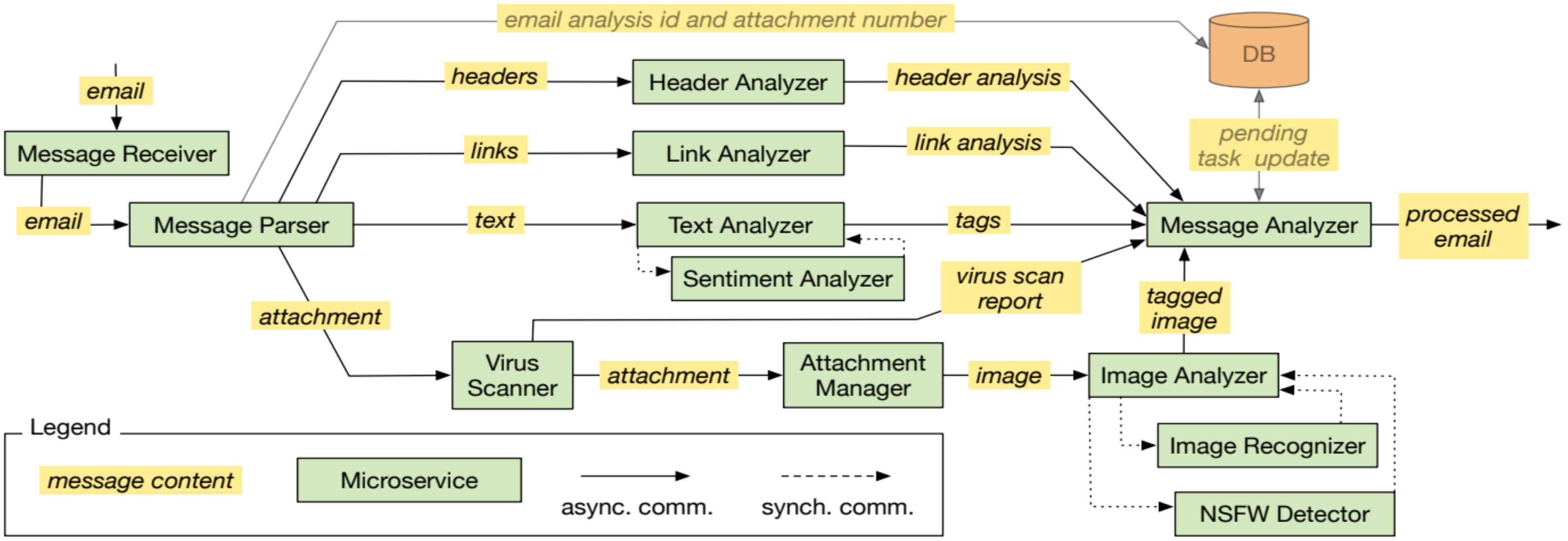
Complexity is not encouraging

.. but ..

- We can assume, due to limited resources and capacity constraints, that the orchestration size is **polynomial** (not exponential)

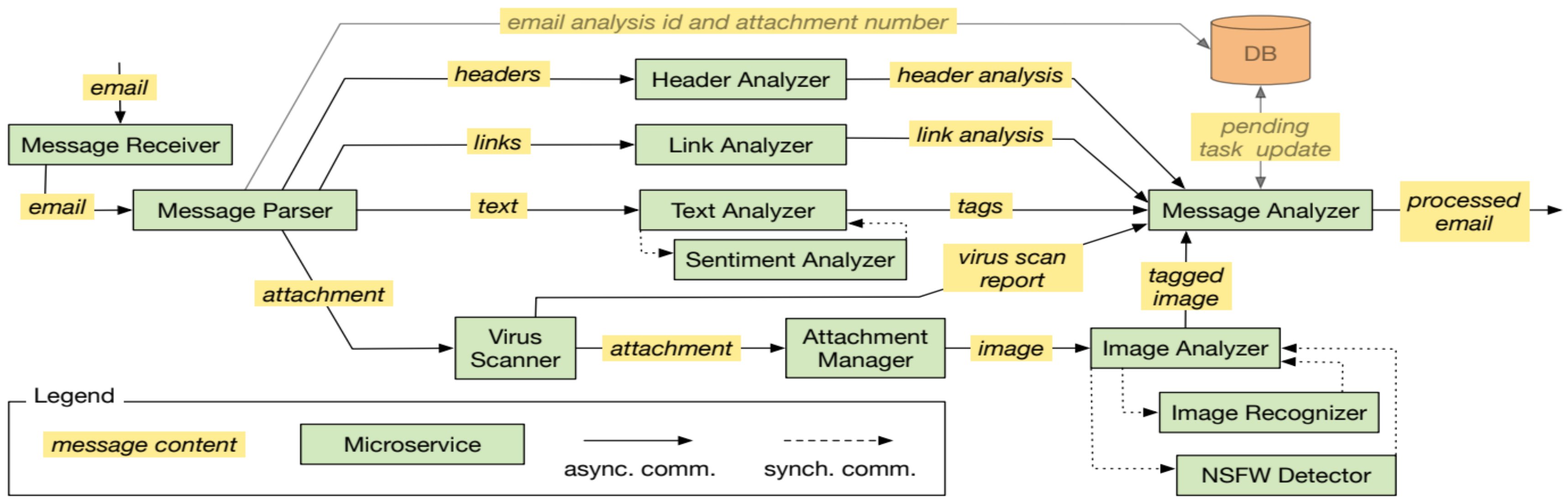
Experimental validation

- We have modeled:
  - a **real-world** microservice architecture
  - computed optimal **deployment** and scaling orchestrations



Experimental validation

- Components/orhcestrations specified in **ABS** (Abstract Behavioural Specification language) executed with Erlang Backend
- Optimal deployments computed by using **SmartDepl** and **Zephyrus2**



ABS feature

- Thanks to ABS expressiveness we have modeled the system including explicit modeling of load balancers
- We have exploited Erlang Backend to execute our simulations
- We have exploited probabilistic properties to evenly distributed email's elementes
- We have exploited ABS time model to observe system's behavior over time
- 1 ABS time unit = 0.005 ms

SmartDepl & Zephyrus2

- **SmartDepl** is a tool to automatically generates ABS deployment code and
- **Zephyrus2** is the engin

System modeling

- **Explicit Request queues** of a fixed maximal size in order to prevent system from over-loading
- Deployment component’s speed **adjusted** at run-time to reflect unused cores

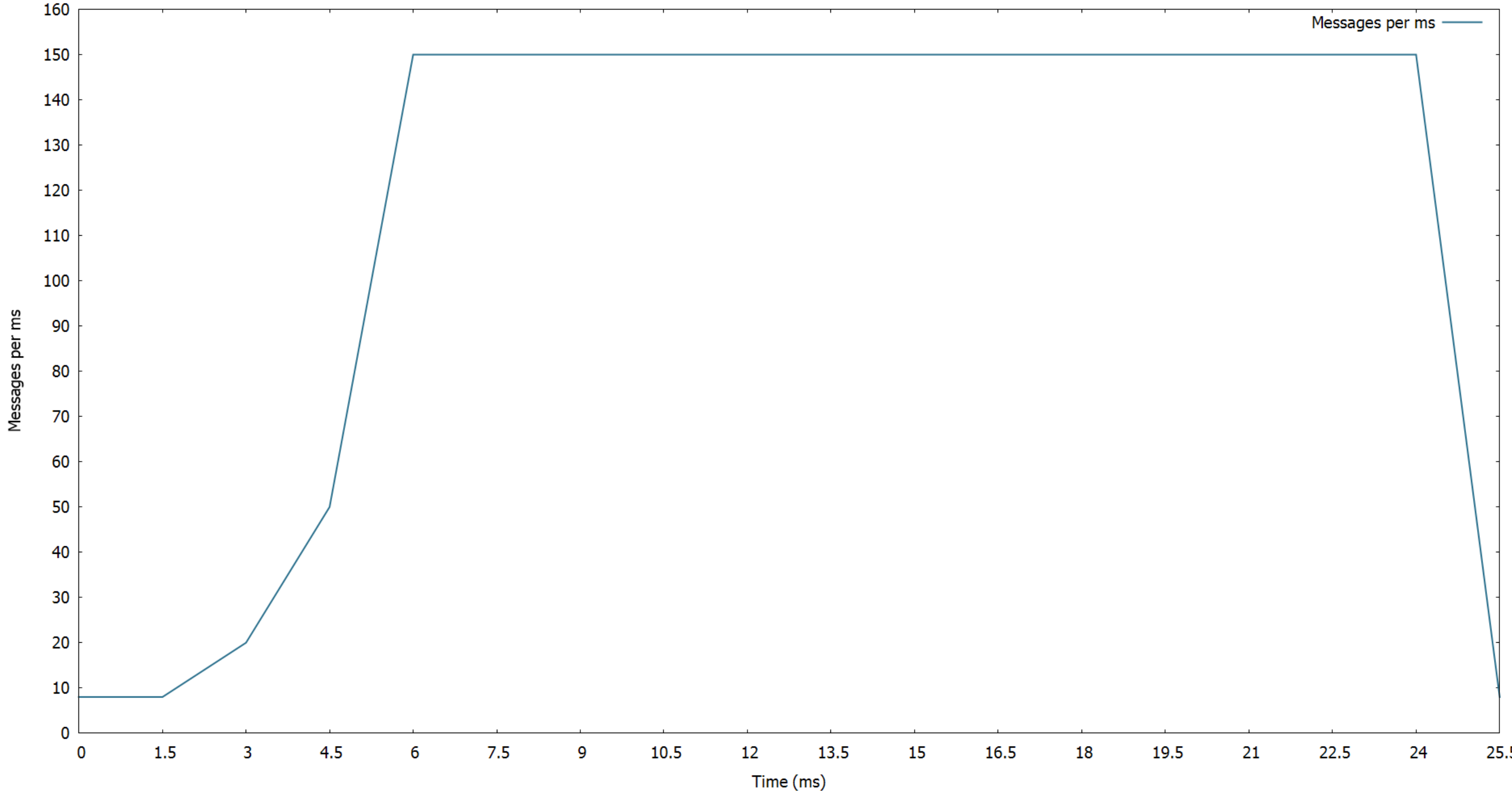
Automatically computed orchestrations

Microservice	Base (10K/sec)	+20K/sec	+50K/sec	+80K/sec
Message Receiver	1	+0	+0	+0
Message Parser	1	+0	+1	+1
Header Analyser	1	+0	+1	+1
Link Analyser	1	+0	+1	+1
Text Analyser	1	+1	+2	+2
Sentiment Analyser	2	+3	+5	+5
Virus Scanner	2	+3	+5	+4
Attachment Manager	1	+1	+2	+1
Image Analyser	1	+1	+2	+1
NSFW Detector	1	+2	+4	+3
Image Recognizer	1	+2	+4	+3
Message Analyser	1	+1	+3	+2

- Models and orchestrations available at:

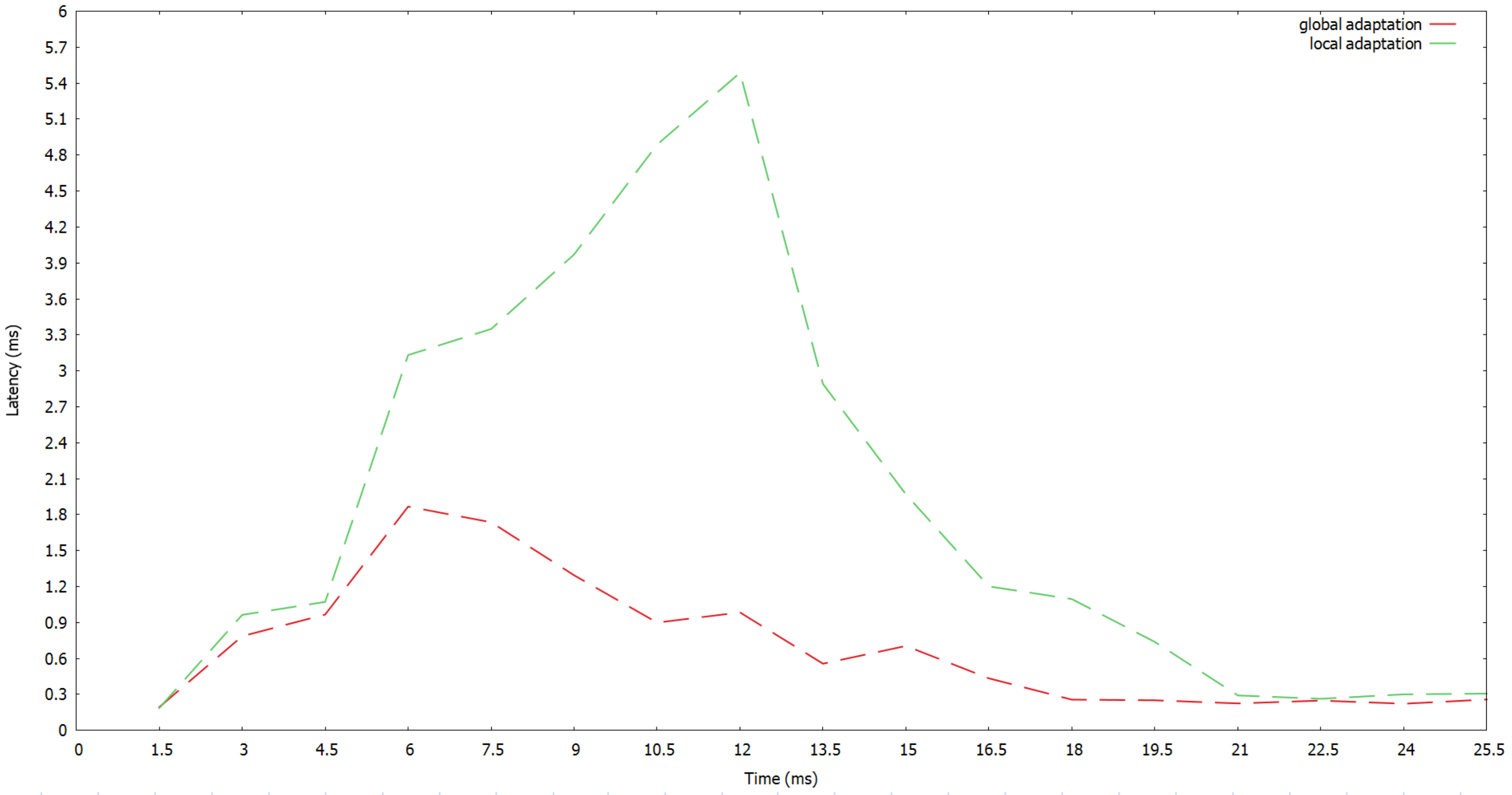
  https://github.com/LBacchiani/ABS-Simulations-Comparison

Scalability experiment

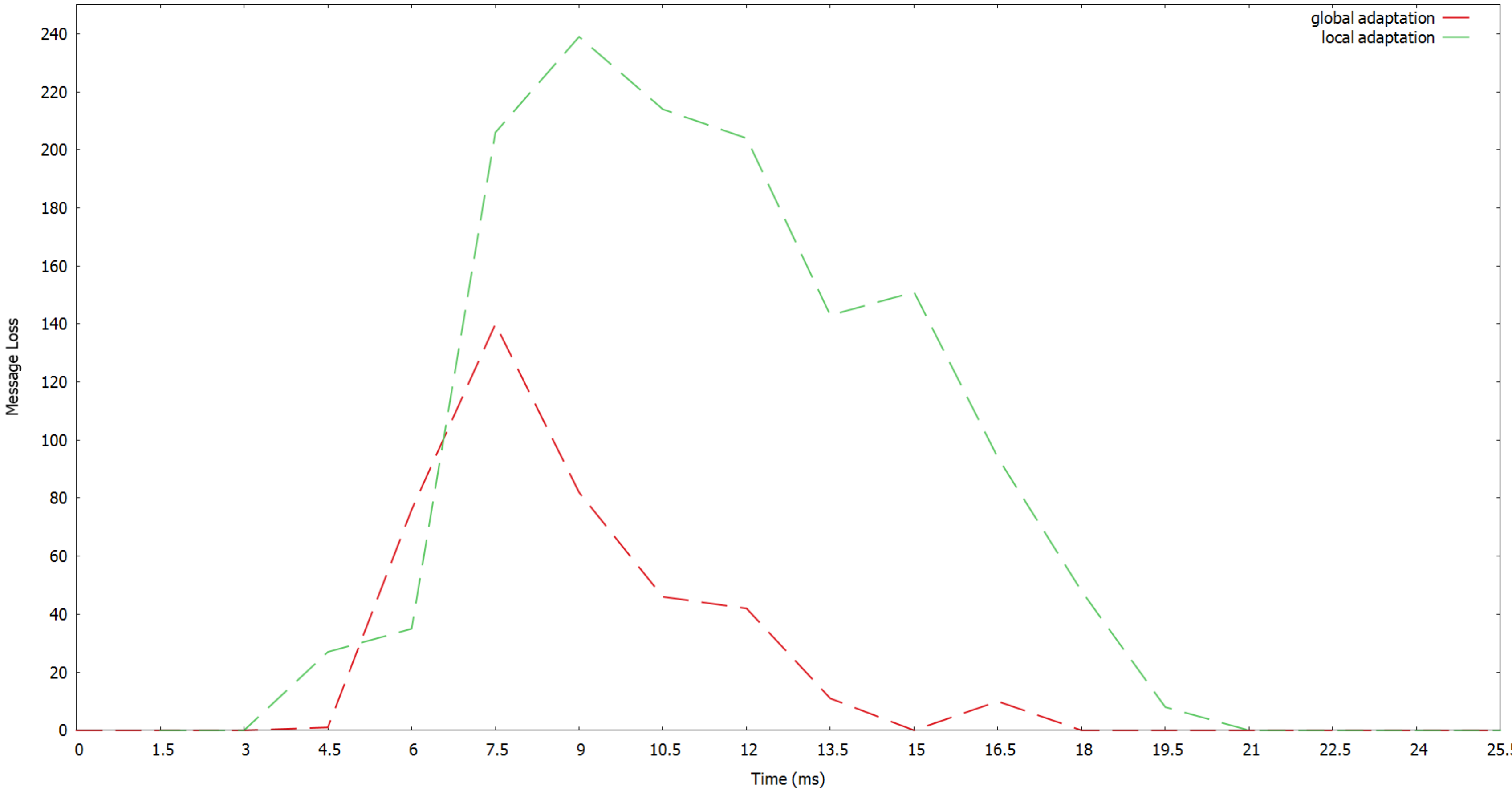


- Message flow grows until it reaches a stable situation

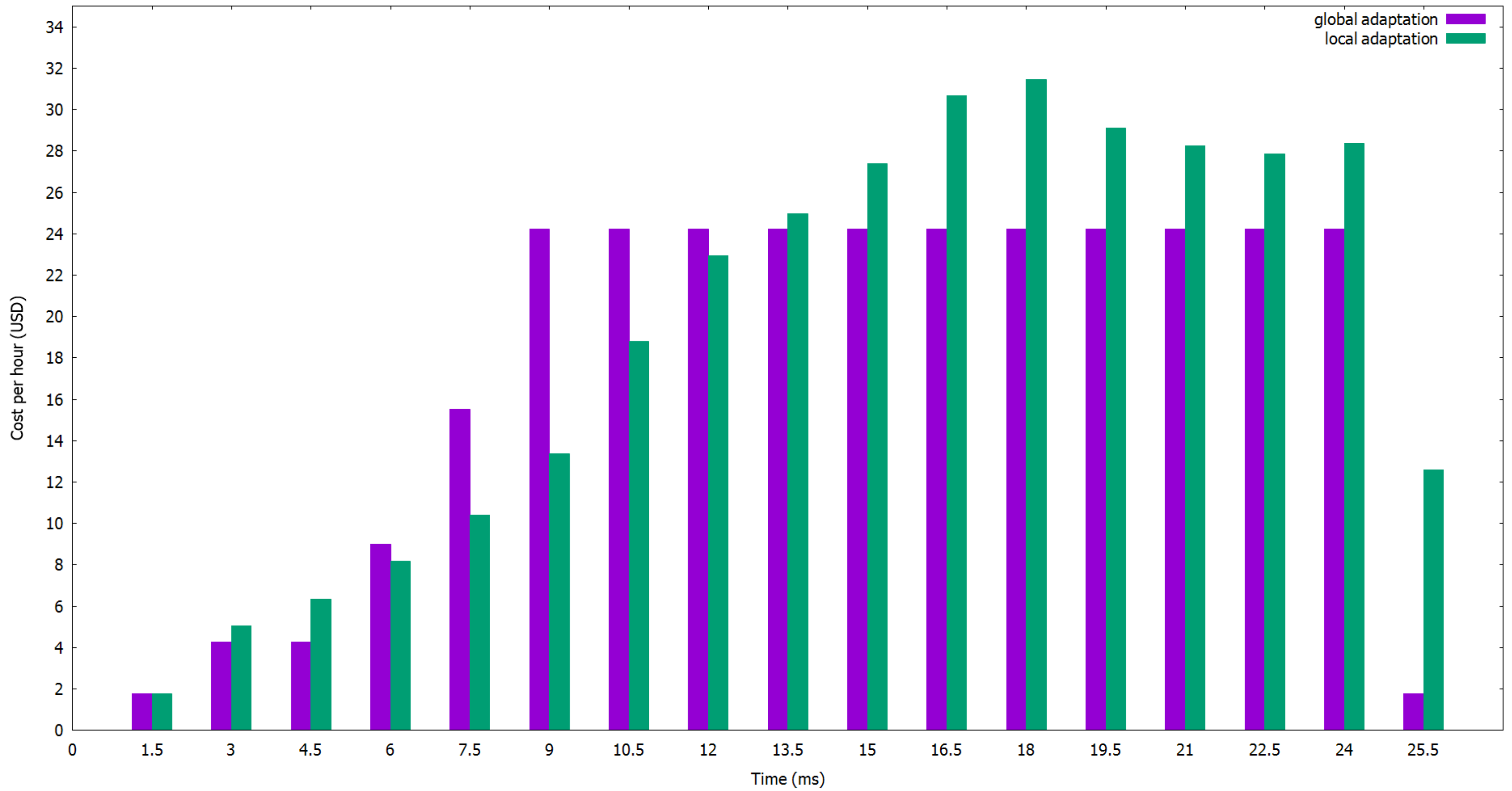
Results are encouraging



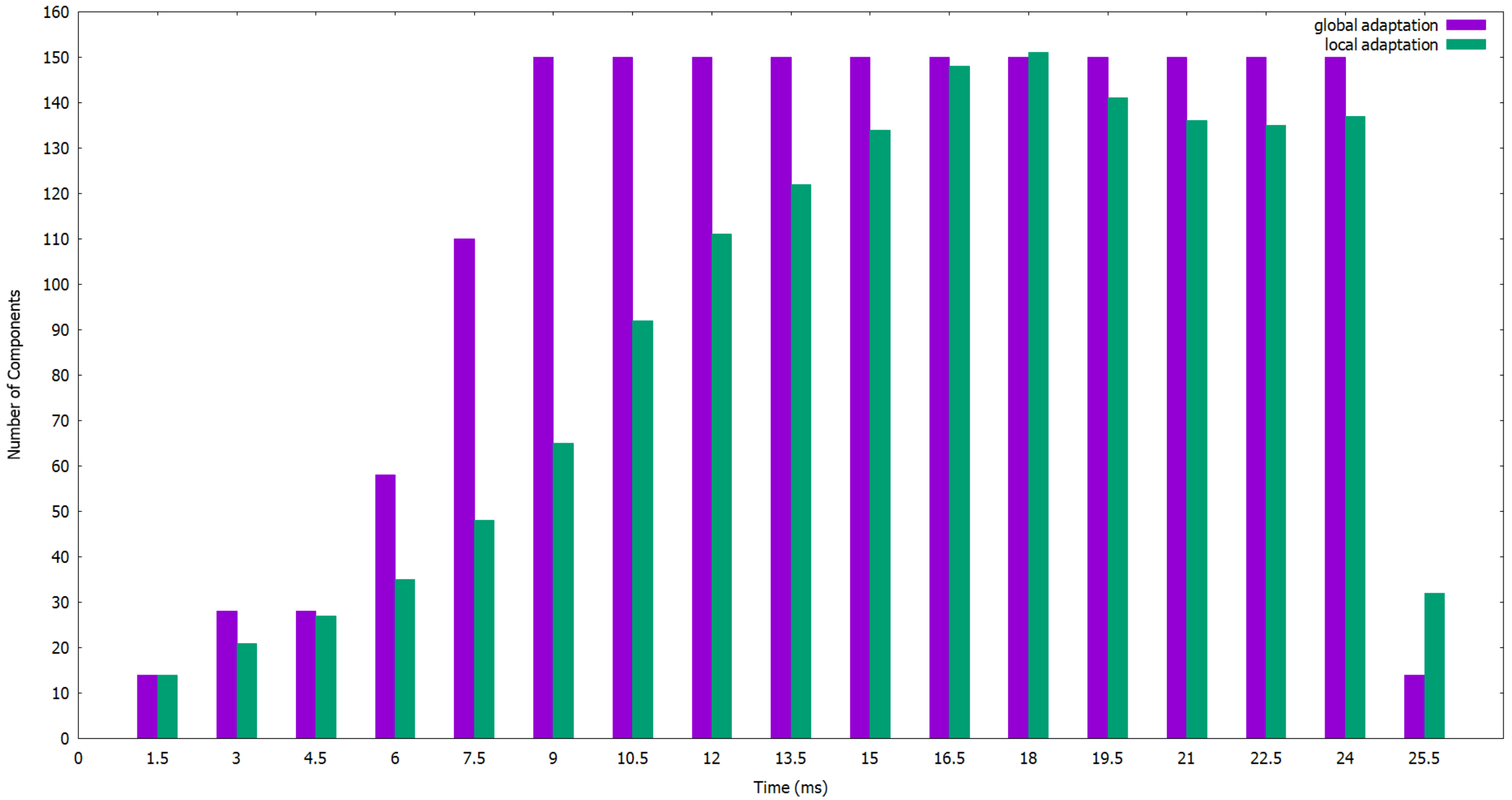
Results are encouraging



Results are encouraging



Results are encouraging



Conclusion & Future work

- (Optimal) deployment of microservice architectures is **decidable** and **fully automatable**
- Our approach **has outperformed** the classic one

Conclusion & Future work

- Future work:
  - **On-line** computation of deployment orchestrations (relax optimality to reduce computation time)