Towards **Network-aware** **Service Composition** in the **Cloud**

WWW 2012

Adrian Klein, Fuyuki Lshikawa, Shinichi Honiden

The University of Tokyo

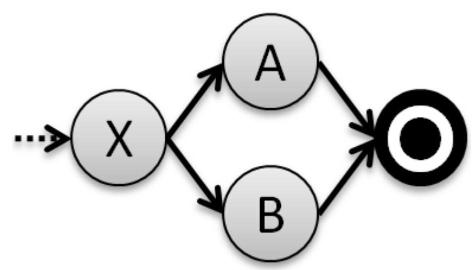
Presented by Jieming Zhu

Slides modified from:
http://www.adrianobits.de/pub/2012_presentation_www2012_adrian_klein.pdf

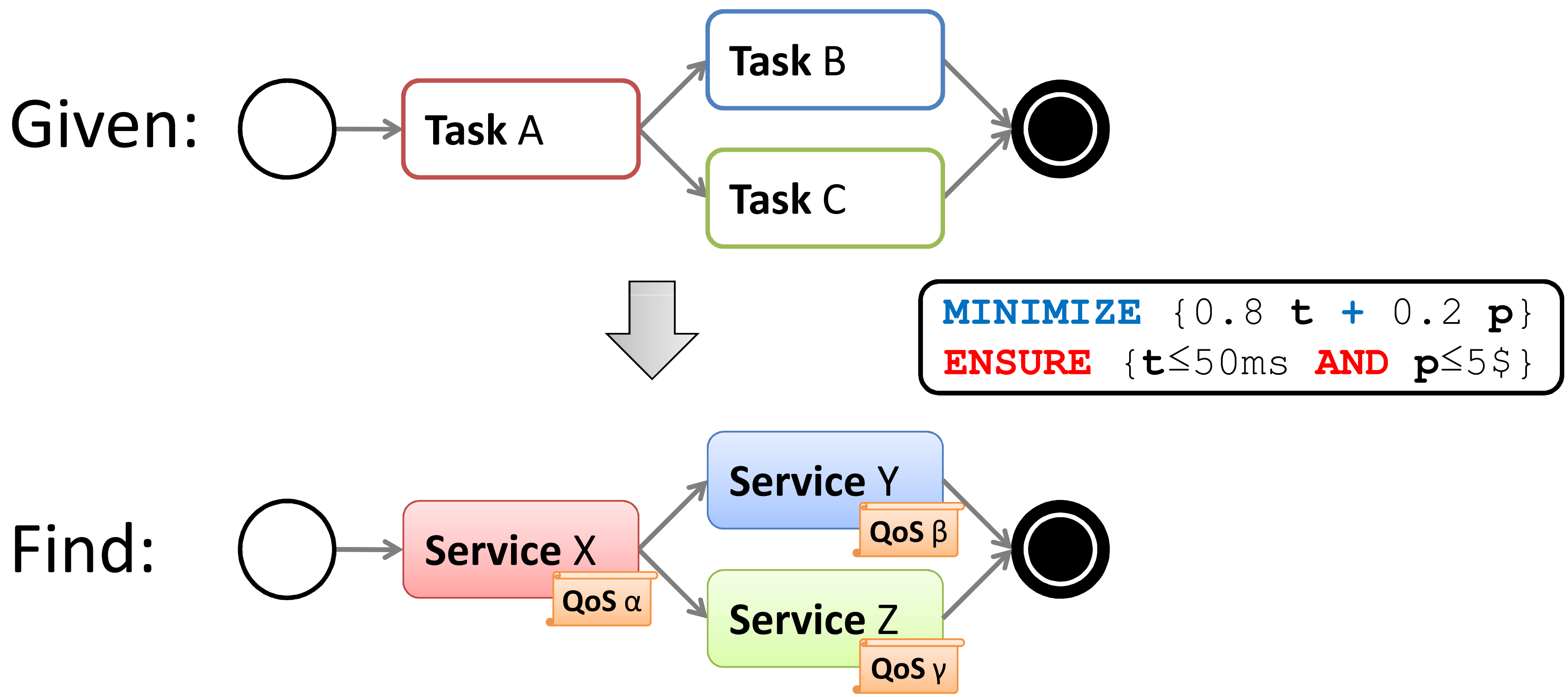
Outline

1. Introduction
2. Approach
3. Evaluation
4. Future work

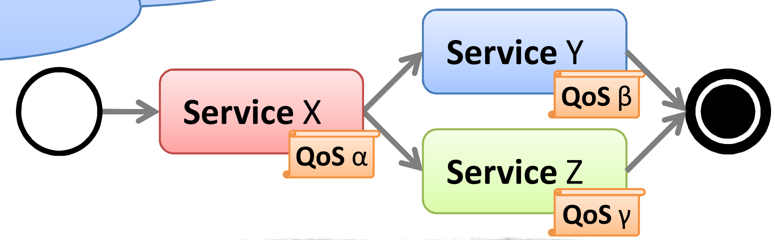
1. INTRODUCTION



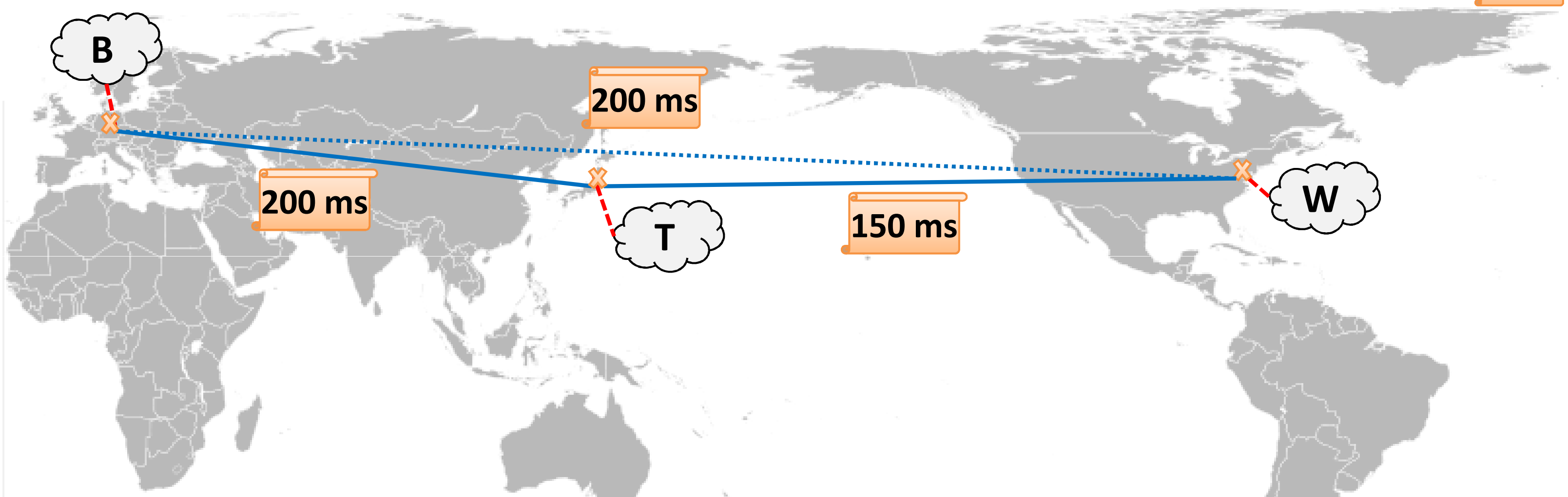
(Standard) Service Composition



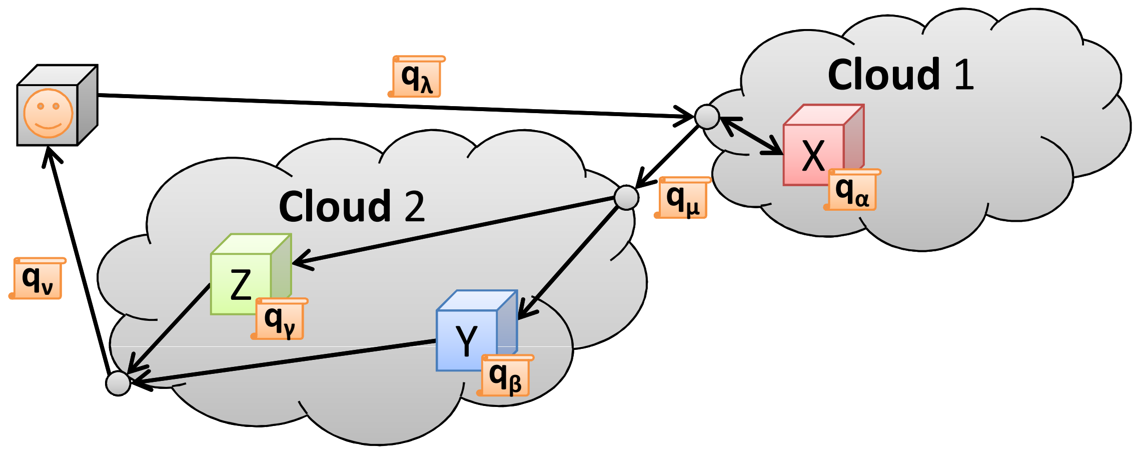
Service Composition in the **Cloud**



Clouds: T, B, W



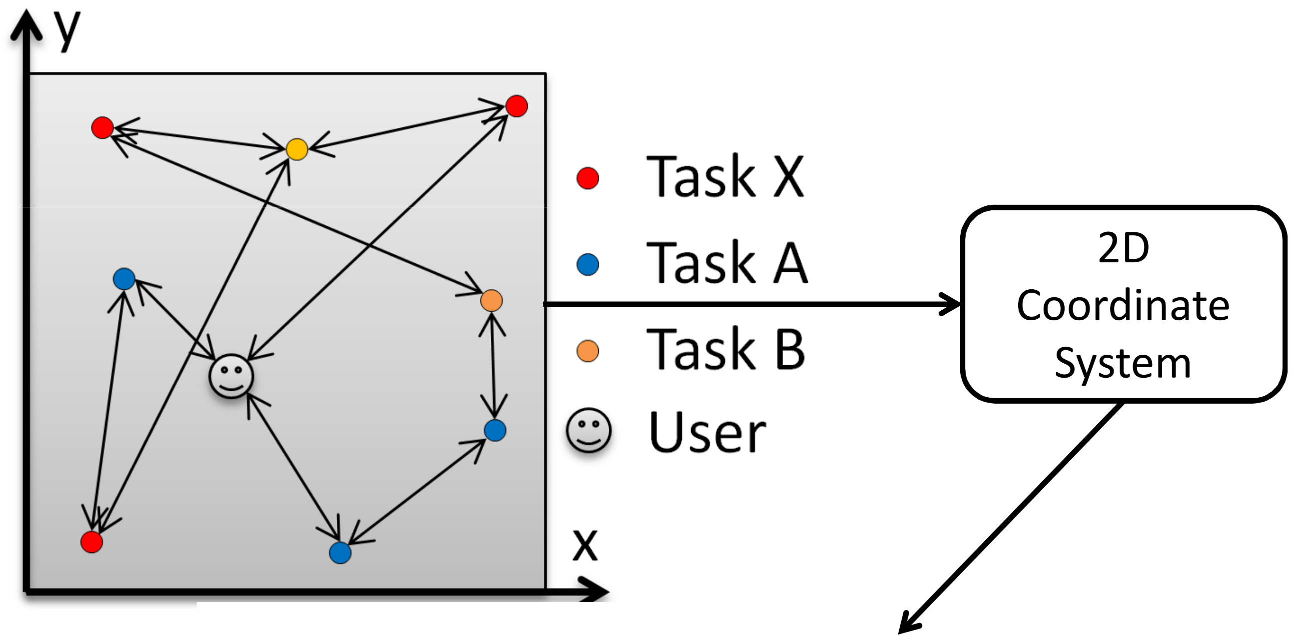
Clouds:	T (Tokyo)	B (Berlin)	W (Washington)
Services for X:	S_1: 500ms	S_2: 90ms	S_3: 80ms
Services for Y:	S_4: 90ms	S_5: 100ms	S_6: 140ms
Services for Z:	S_7: 110ms	S_8: 100ms	S_9: 110ms



TWO CHALLENGES…

1. Challenge: **Network-Awareness**

How to **compute** the latency between **arbitrary** two services/users?

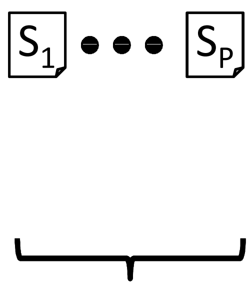


All-Pairs Ping too costly => Estimate w. **Network Model**!

total: O(**N**^{2}), add: O(**N**)   total: O(**N**), add: O(**1**)

2. Challenge: **Scalability**

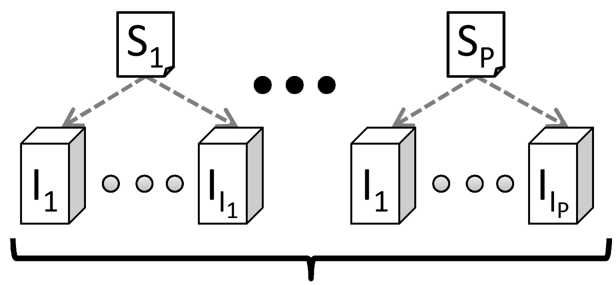
- Before
  - P providers for Task T



P choices per Task

=> **[50-100]** choices (per Task)

- Cloud
  - P providers for Task T
  - I_k **cloud instances** per P_k



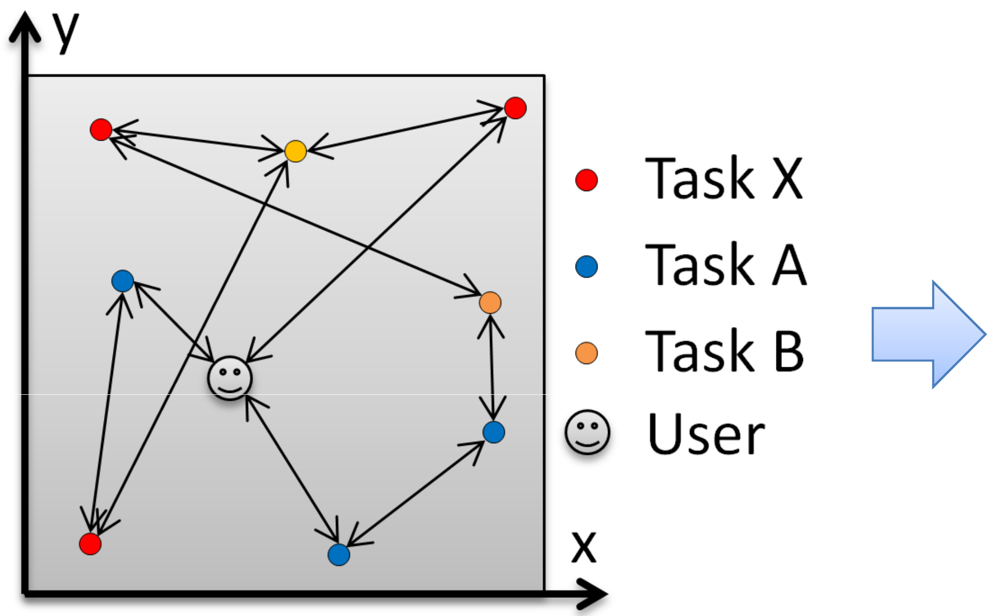
$P \cdot \sum I_k$ choices per Task

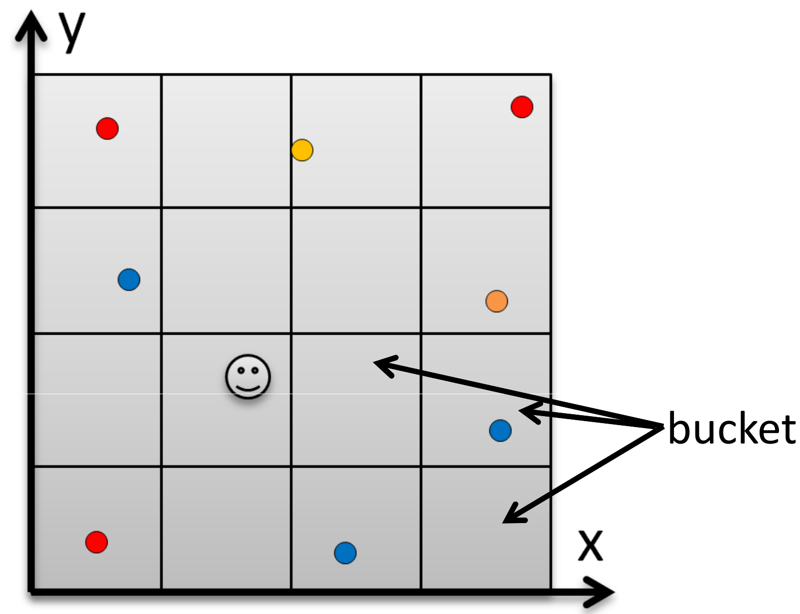
=> [50-100]x[20-120] = **[1000-12000]** choices (per Task!)

2. APPROACH

- a) **Network** Model
- b) **Network-aware** QoS Computation
- c) **Network-aware** Selection Algorithm

a) Network Model





Build from existing model

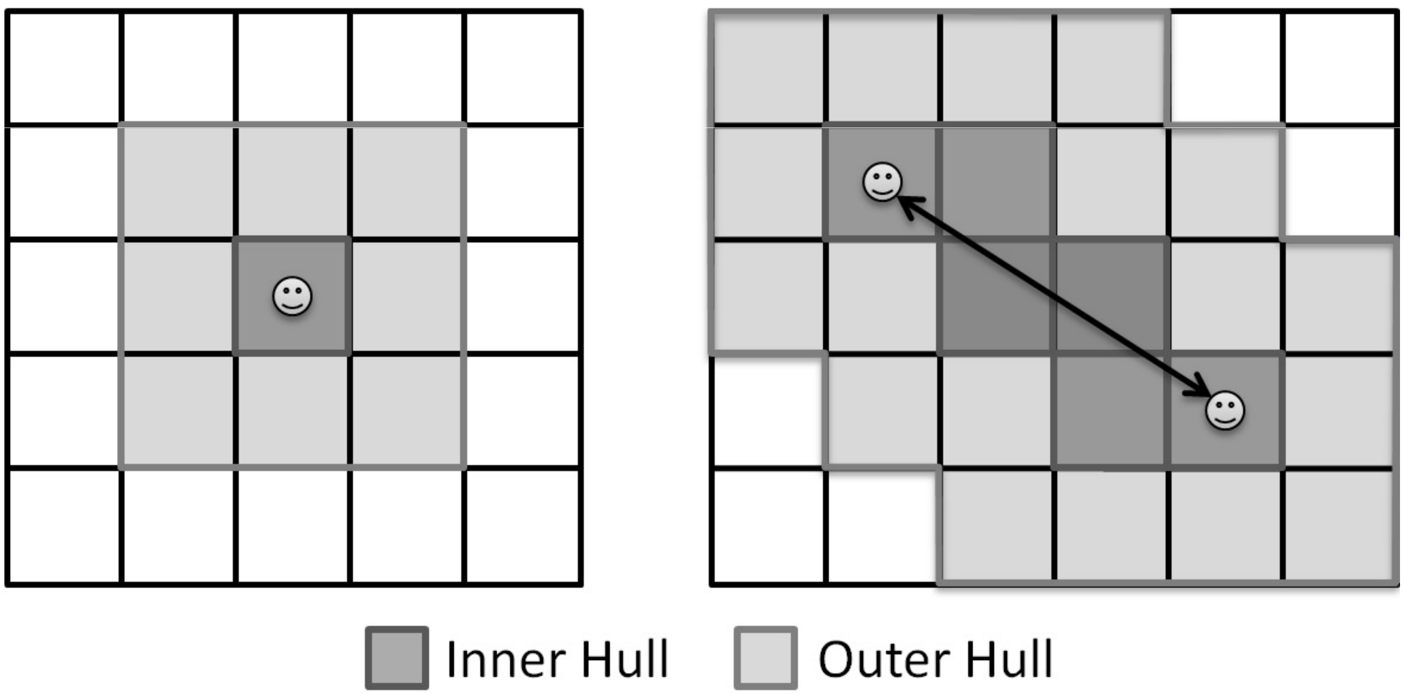
(2D-Coordinate System, e.g. Vivaldi)

Build on top of model

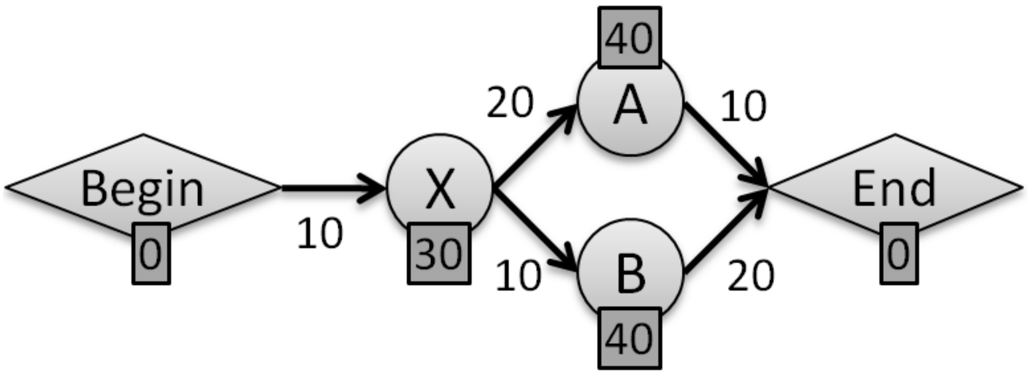
(Hash into **buckets**)

Operations on *Network Model*

- Compute **Hull** of a network **location**
- Compute **Hull** of a network **path**



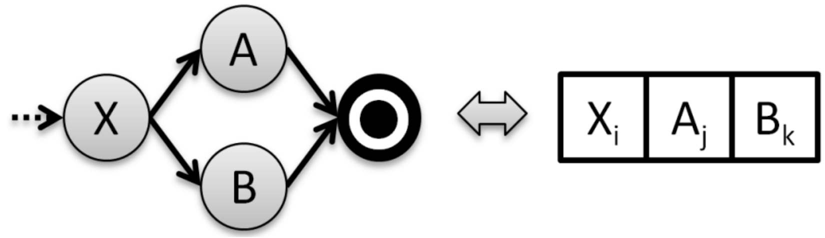
b) Network-aware QoS Computation

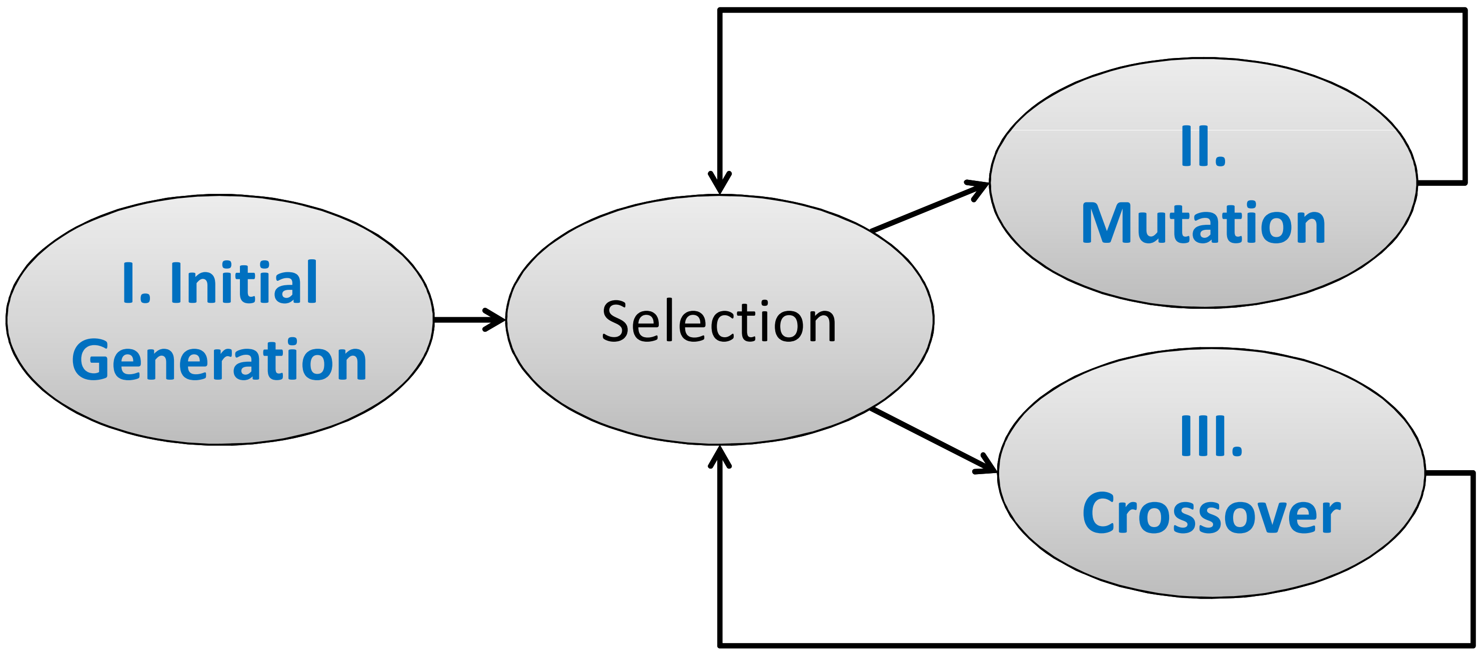


#	Begin	X	A	B	End
0	0 / ?	0 / ?	0 / ?	0 / ?	0 / ?
1	**0 / 0**	10 / 0	0 / ?	0 / ?	0 / ?
2	0 / 0	**10 / 40**	60 / ?	50 / ?	0 / ?
3	0 / 0	10 / 40	**60 / 100**	50 / ?	110 / ?
4	0 / 0	10 / 40	60 / 100	**50 / 90**	110 / ?
5	0 / 0	10 / 40	60 / 100	50 / 90	**110 / 110**

c) Network-aware Selection Algorithm

Custom Genetic Algorithm



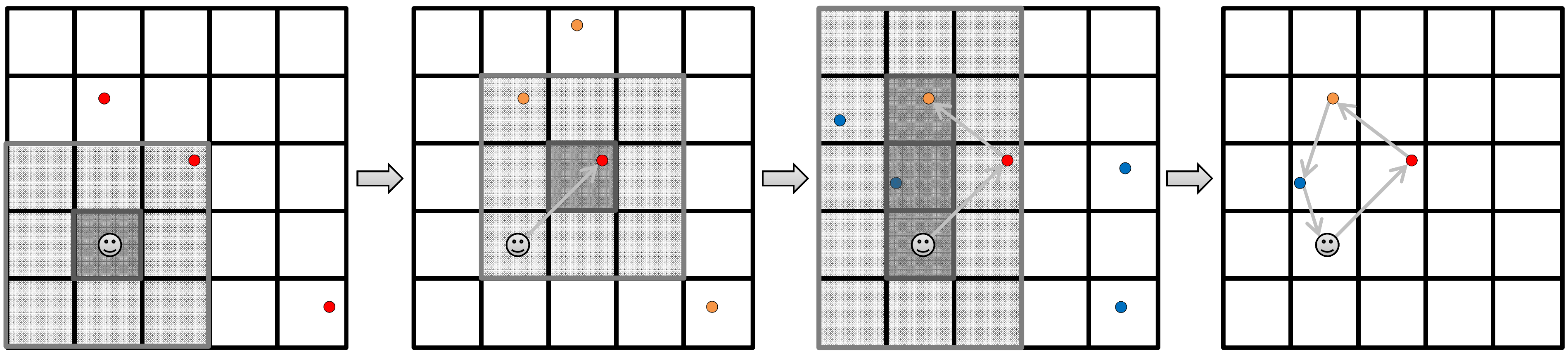


I. Initial Generation

Localizer Heuristic:

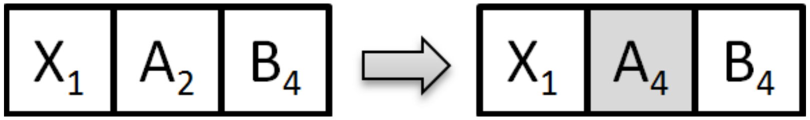
Find workflows with low latency

by choosing "close" service in each step.



II. Mutation

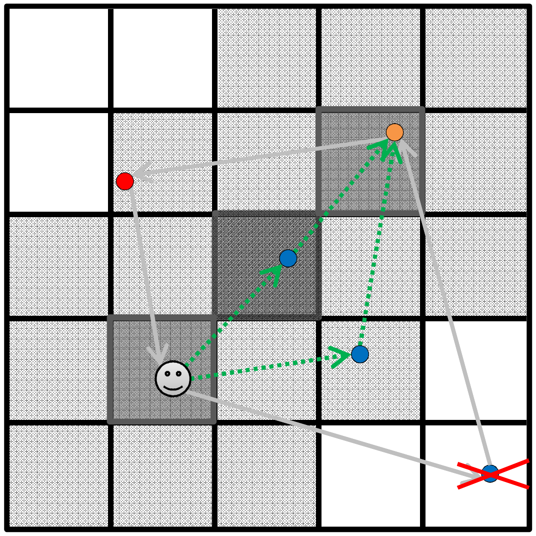
Classic:



Mutate some places randomly

Localizer Heuristic:

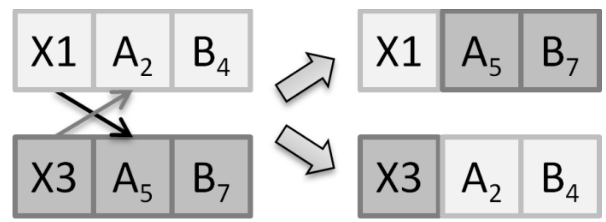
Choose “close”services
for some places randomly



III. Crossover

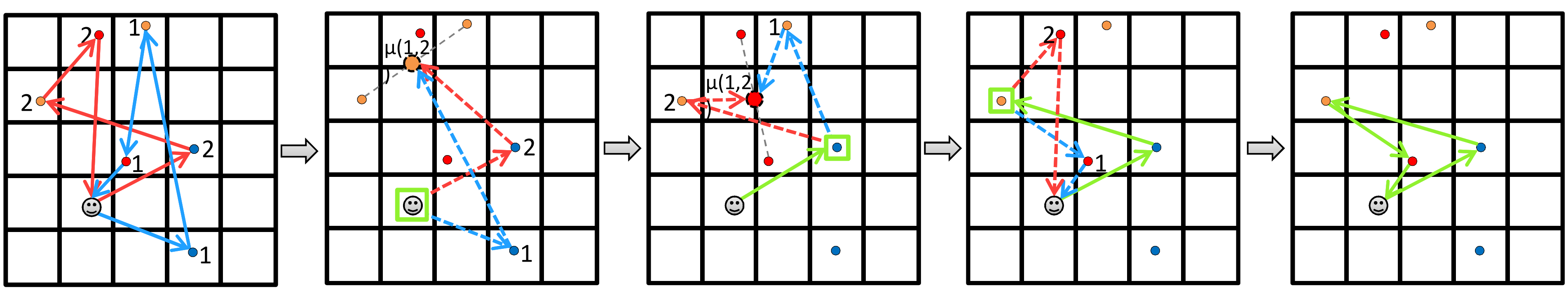
Classic:

Interchange at a number of points (1, 2, …, N)



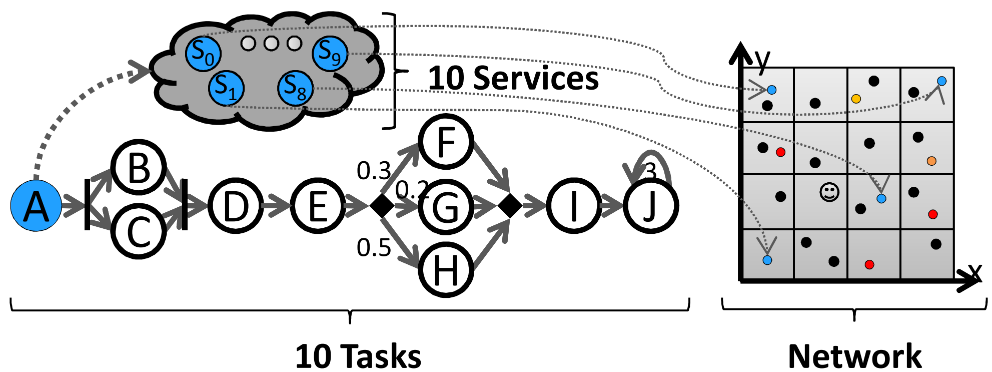
Localizer Heuristic:

Choose “closer”services from parents
(randomly in proportion to their distance from “last” and “next” service)



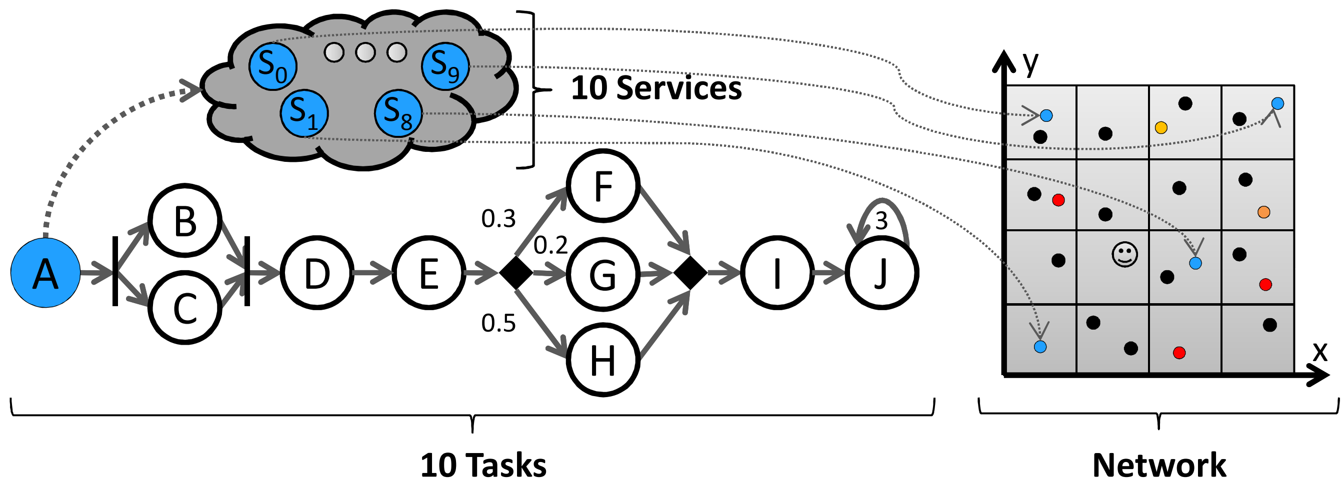
Parents 1 & 2 | Choose 1. Service | Choose 2. Service | Choose 3. Service | Final Offspring

3. EVALUATION



Setup

Randomly generate:

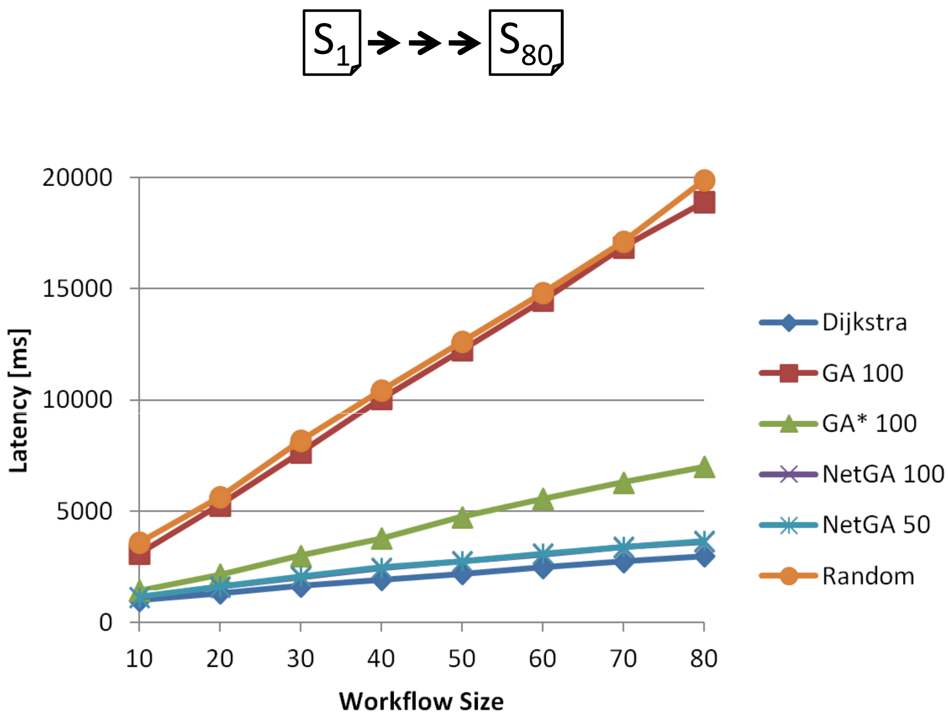


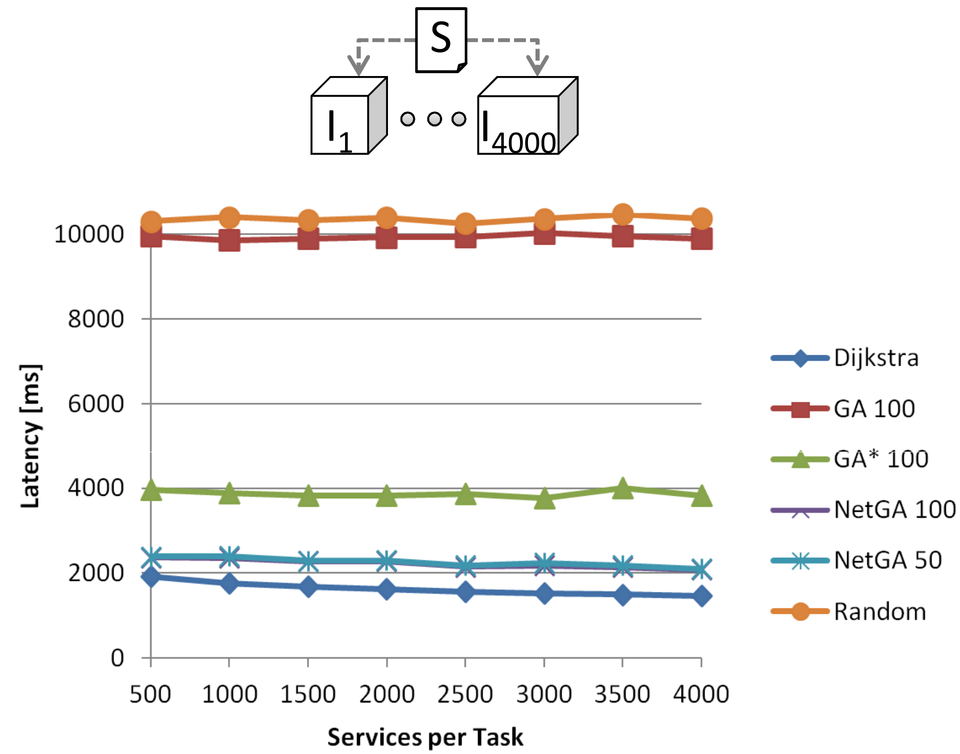
Workflow sizes:	10 – 80 (normal)
Services per task:	500 – 4000 (quite a lot! normally < 500)
QoS values:	at random from uniform distr.
Number of Evaluations:	512 test cases for each data point

Algorithms

- Random (baseline)
- GA 100 (standard approach, population of 100)
- GA* 100 (st. appr. augmented w. Network Model)
- ~~GA* 50 (pop. of 50)~~ *solutions too bad*
- NetGA 100 (our full approach)
- NetGA 50 (our full approach, pop. of 50)
- Dijkstra (optimum)

End-user **Latency** [of exec. workflows]

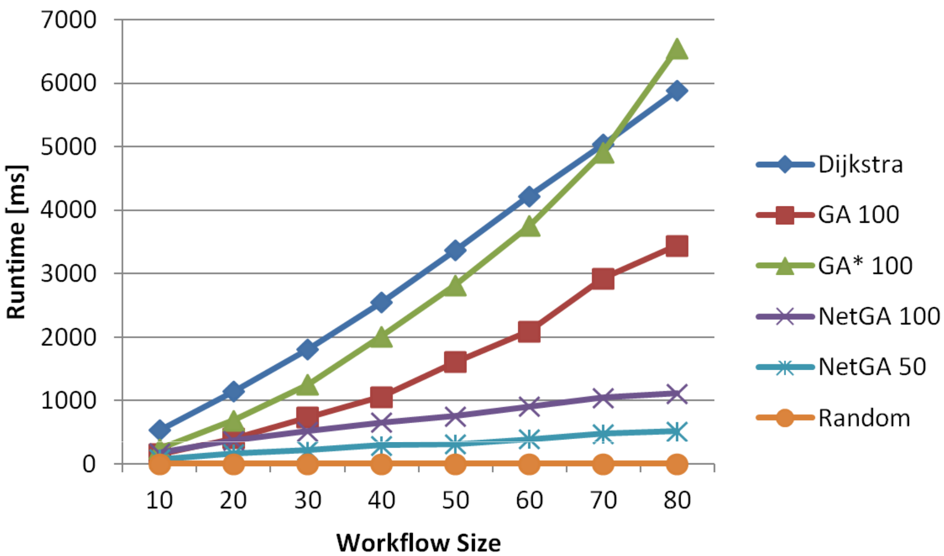


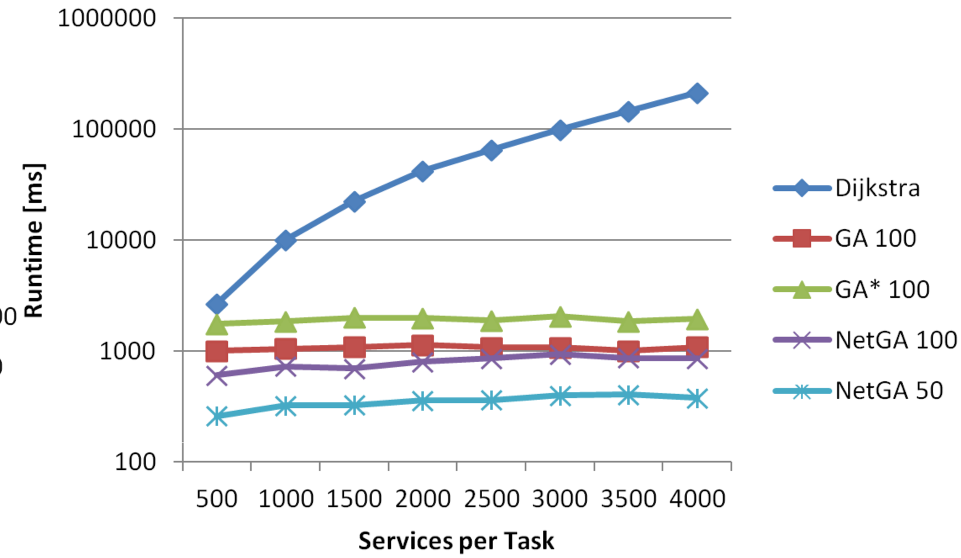


End-user Latency = Network Latency **+** Execution Time

∑Execution Time was fixed to 1000 ms for all workflows.

Optimization **Runtime** [of algorithms]

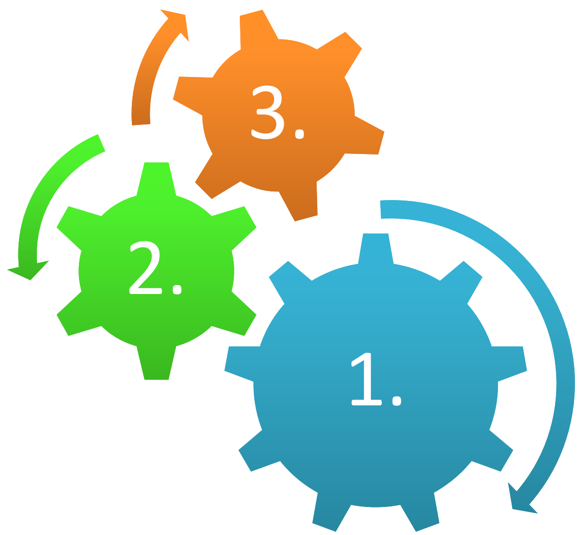




Algorithm	a	o(t)	Θ(t)	O(t)
Dijkstra	32	$n^{1.18}$	$\mathbf{n^{1.19}}$	$n^{1.22}$
GA 100	2.7	$n^{1.62}$	$n^{1.67}$	$n^{1.77}$
GA* 100	3.4	$n^{1.71}$	$\mathbf{n^{1.72}}$	$n^{1.84}$
NetGA 100	17	$n^{0.95}$	$n^{0.97}$	$n^{1.03}$
NetGA 50	10	$n^{0.87}$	$\mathbf{n^{0.90}}$	$n^{0.92}$

Approximation $a \cdot n^x$ with minimal square error

3. FUTURE WORK



Future Work

1. Multiple QoS
   – **Evaluate** if standard GA **beats** us
     when **latency** is not so important
2. Real Data
   – **Analyze** PlanetLab Traces (recorded data)
     => build prediction model with Vivaldi
   – **Verify** that results are as accurate as
     the latency prediction (**≤ 10-15%**)
3. GA Operators
   – **Evaluate** different variations in more detail…

Thank you!

Q & A