Towards scalable deployment optimization in the Fog using MDPs and Function Approximation

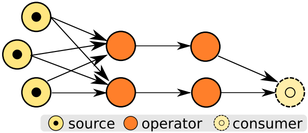
Gabriele Russo Russo

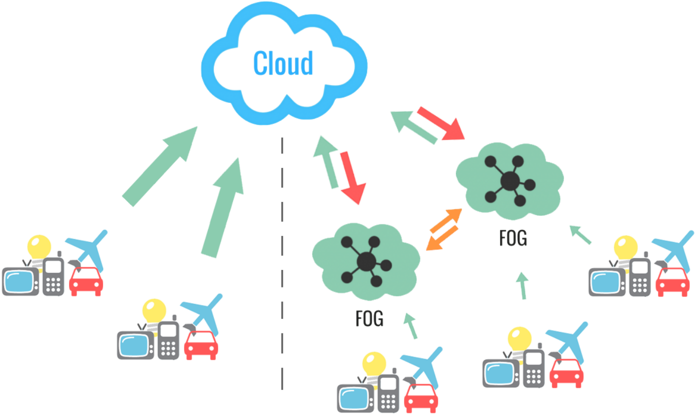
University of Rome Tor Vergata

Università di Roma Tor Vergata

Data Stream Processing (DSP)

processing Big Data in real-time





New trend: moving computation towards data sources and consumers

Geo-distributed DSP: old and new challenges

- Non negligible network latency
- Heterogeneous computing resources (and usually less powerful...)
- Variable infrastructure conditions

$\Downarrow$

- Application deployment must be adapted at run-time:
  - how many parallel replicas for each operator? (**elasticity**)
  - where to deploy each operator?
  - when to change the deployment, incurring overhead?

Operator deployment adaptation

Operator Elasticity

- Parallelism should change over time depending on input data rate

Heterogeneous infrastructure

- Computing infrastructure composed of regions
- Several types of computing resources available (e.g., VMs with different capacity)

Operator deployment adaptation

Operator Elasticity

- Parallelism should change over time depending on input data rate

Heterogeneous infrastructure

- Computing infrastructure composed of regions
- Several types of computing resources available (e.g., VMs with different capacity)

Operating costs for a single operator

- resources cost: depends on amount and type of used resources
- adaptation cost: proportional to performance degradation at each deployment reconfiguration
- SLA violation: paid whenever the performance (i.e., processing latency) violates a given threshold

→ would like to minimize all of them in the long-term

MDP formulation

We model the problem as an infinite-horizon **Markov Decision Process**

- System **state**: current deployment and input data rate
- **Actions**: possible deployment adaptations

- Each state-action pair (s, a) associated with a **cost** $c(s, a)$
- We search for the optimal **policy**:

$$\text{minimize} \quad \sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \qquad \gamma \leftarrow \text{discount factor} \in [0, 1)$$

MDP formulation

We model the problem as an infinite-horizon **Markov Decision Process**

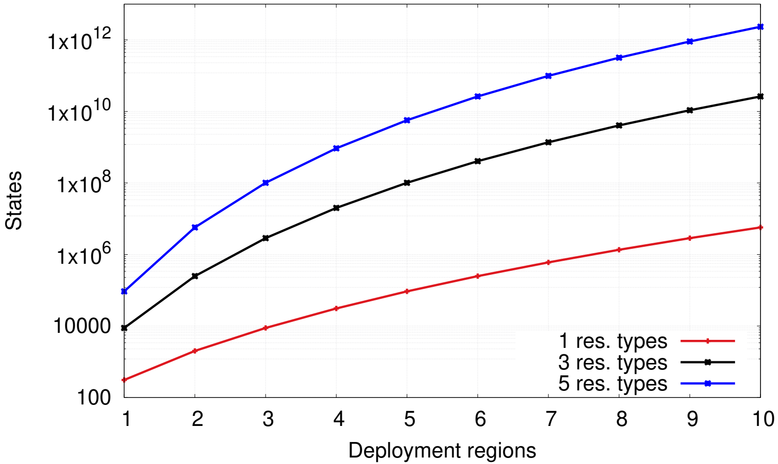
- System **state**: current deployment and input data rate
- **Actions**: possible deployment adaptations

- Each state-action pair (s, a) associated with a **cost** $c(s, a)$
- We search for the optimal **policy**:

$$\text{minimize} \quad \sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \qquad \gamma \leftarrow \text{discount factor} \in [0, 1)$$

- Can be solved by DP, LP, reinforcement learning, ...
- Resolution based on the Q function
- Traditional algorithms store Q in memory: an entry for each state-action pair

Scalability



22 GB of memory to store Q with 5 regions and 3 classes of resources

Does not scale in a Fog scenario (many applications to optimize!)

Function Approximation for MDPs

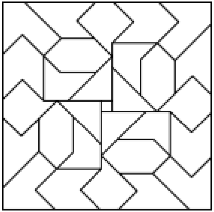
- Idea: replacing the Q table with a parametric function $\hat{Q}(s, a, \theta)$
- Need to store (and compute) only the parameters θ

- Today we focus on Linear Function Approximation:
  $\hat{Q}(s, a, \theta) = \sum_i \phi_i(s, a)\theta_i$
- Defining a good set of features $\phi_i(s, a)$ is challenging
  - More features = more parameters to compute and store
  - A small set of features may prevent the algorithm to converge

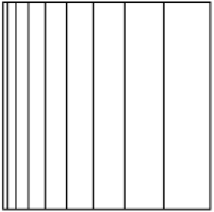
Tile Coding

Idea: cover the state space with "tilings"

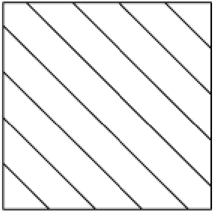
- adjacent states are aggregated in a single tile
- each state activates a tile (i.e., binary feature)
- fine-grained vs. coarse-grained tilings
- different number of dimensions and shape of tiles



a) Irregular



b) Log stripes

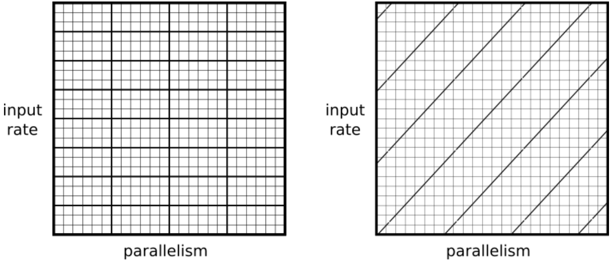


c) Diagonal stripes

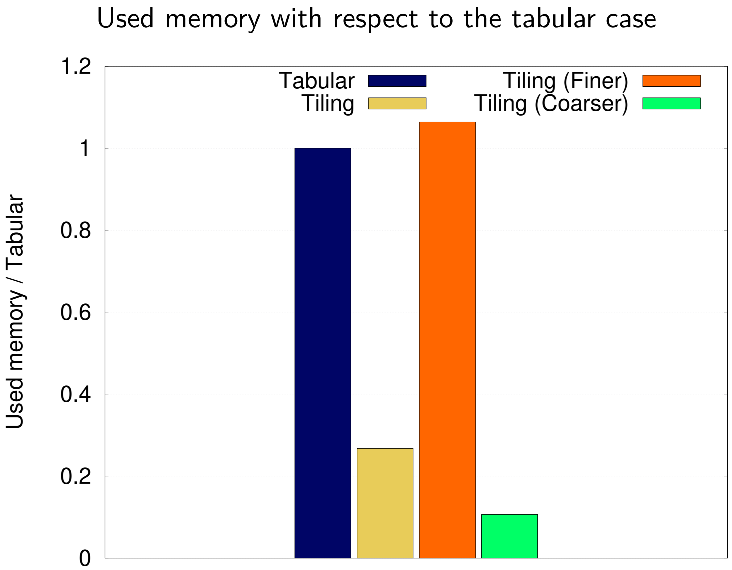
Using Tile Coding

First step: homogeneous computing resources

- A binary feature for scaling operations (scale-out, scale-in)
  $\rightarrow$ captures adaptation cost
- Rectangles-based tilings to group states with similar parallelism and input rate
- A stripes-based tiling to group states with similar load per replica
- 3 granularity settings: base, finer, coarser

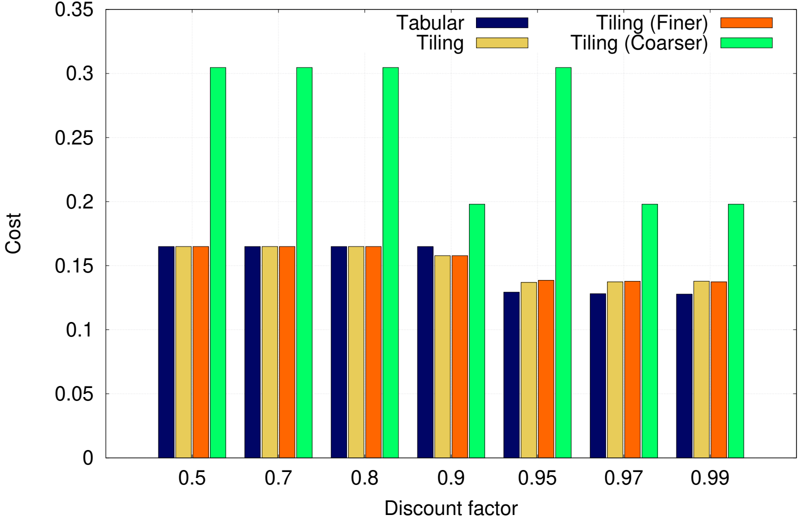


Results: used memory



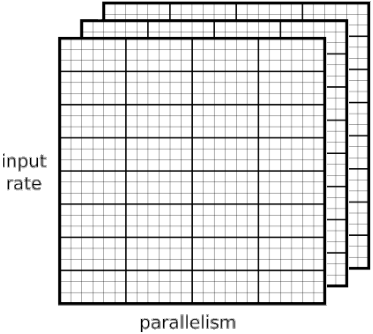
Simulation results: average cost

Base scenario, with homogeneous resources

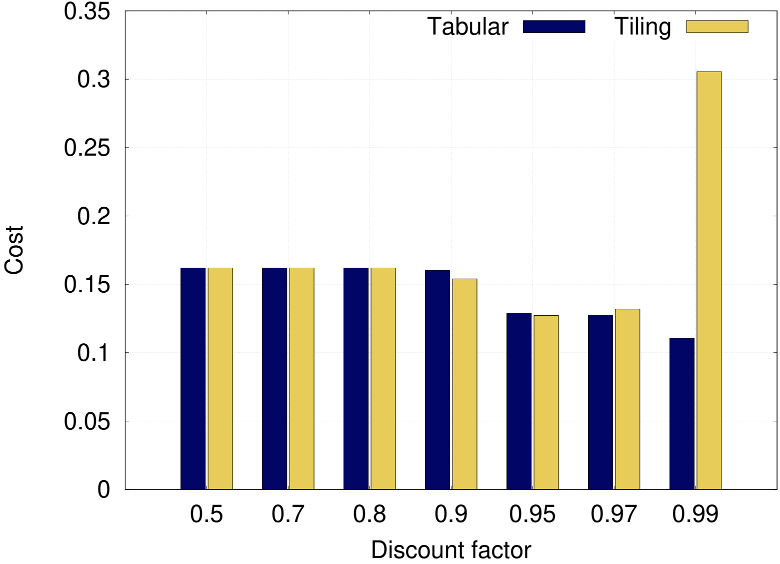


Features for the heterogeneous scenario

- Considering parallelism is not enough any more: computing resources with different cost and performance
- Would need a N-dimensional tiling: input rate + amount of resources of each type
- Simpler idea, adding only a third dimension to the current tilings: parallelism, input rate, type of the less powerful used resource



Preliminary results: 3 types of comp. resources



Near-optimal results for $\gamma < 0.99$, using 2% of the memory

Conclusion

- A MDP-based framework for optimizing deployment in the Fog
- Function Approximation techniques are promising for scalability

Still work to do for better performance:

- Automatic feature engineering (e.g., adaptive tiling)
- Artificial Neural Networks

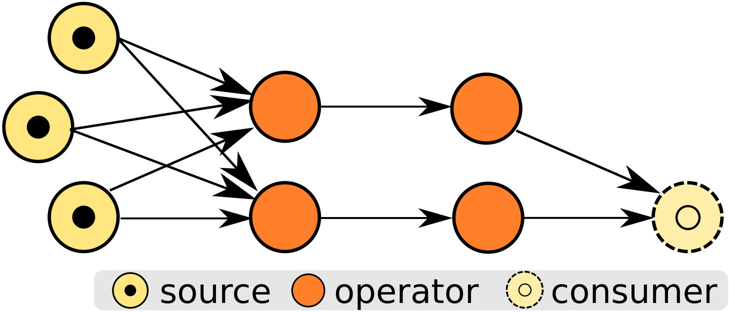
+ Extend to similar resource allocation problems in the Fog

Thanks for your attention!

`russo.russo@ing.uniroma2.it`
`www.ce.uniroma2.it/~russorusso`

Data Stream Processing (DSP)

- A computational paradigm for **real-time Big Data analysis**
- Continuous processing of unbounded sequences: data streams
- Data processed "on the fly"



MDP formulation

We model the problem as an infinite-horizon **Markov Decision Process**

- System **state**: $s = (\mathbf{K}, \lambda)$
  $k_{r,\tau}$ = replicas deployed in region r using instances of type τ
  λ = current input data rate for the operator (discretized)
- **Actions**:
  - add a replica on a resource of type t in region r
  - kill one of the active replicas
  - do nothing
- Each state-action pair associated with a **cost** $c(s, a)$:
  $c(s, a) = w_R c_{resources}(s, a) + w_A c_{adaptation}(s, a) + w_S c_{SLA}(s, a)$
- We search for the optimal **policy** $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$:

$$\text{minimize} \quad \sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \qquad \gamma \leftarrow \text{discount factor} \in [0, 1)$$

Solving the MDP

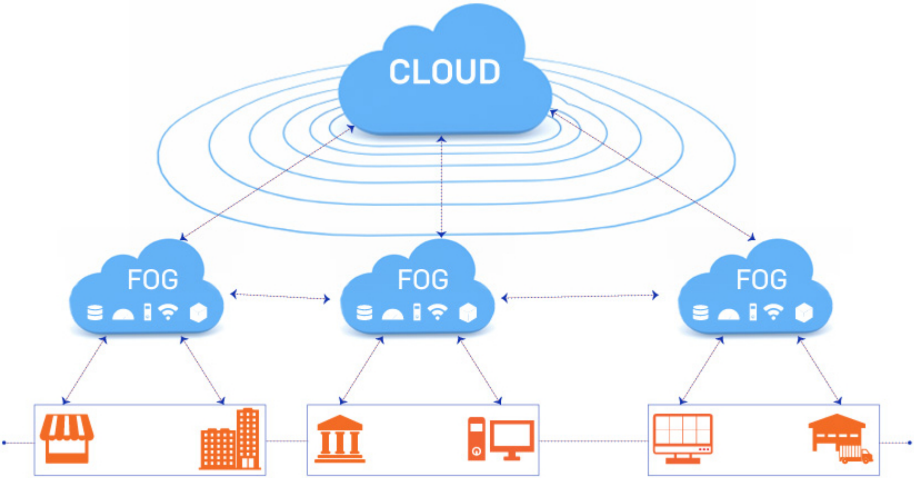
<1-| handout:0> An optimal policy can be found by standard techniques:
linear programming, dynamic programming, reinforcement learning, ...

- Classical DP algorithms (e.g., Value Iteration) rely on **Q function**: expected long-term cost of every action in every state
- Computed iteratively until convergence
- Q function stored in a Q table in memory: an entry for each state-action pair

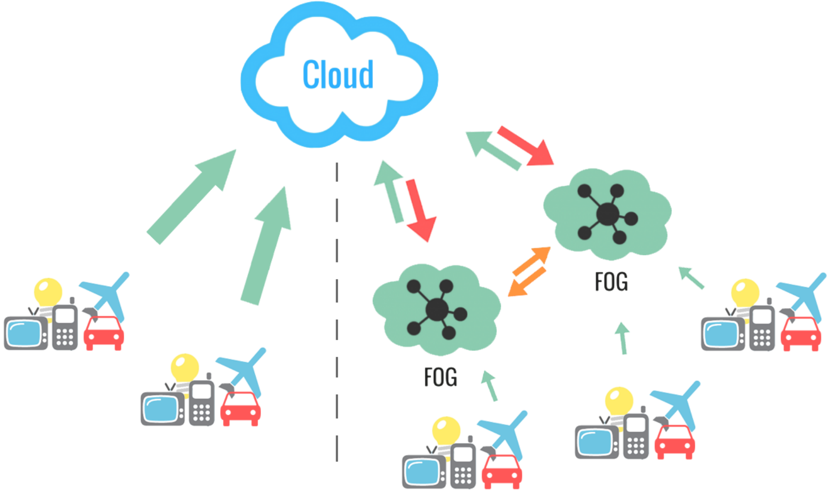
Trajectory Based Value Iteration

Algorithm 3: Trajectory Based Value Iteration (TBVI)	Complexity
Input: MDP, α, L_1	
Output: π	
$\boldsymbol{\theta} \leftarrow$ Initialize arbitrarily	
while *time left* **do**	
for $\langle s, a \rangle$ *in a trajectory following* π^ϵ **do**	
Create L_1 samples: $s'_j \sim \mathcal{P}^a_{s\cdot}, j = 1, ..., L_1$	
$Q^+(s, a) \leftarrow \frac{1}{L_1} \sum_{j=1}^{L_1} \mathcal{R}^a_{ss'_j} + \gamma \max_{a'} Q(s'_j, a'),$	$\mathcal{O}(nL_1\|\mathcal{A}\|)$
$\delta \leftarrow Q^+(s, a) - Q(s, a)$	
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\phi(s, a)$	$\mathcal{O}(n)$
return π greedy with respect to Q	

Fog Computing



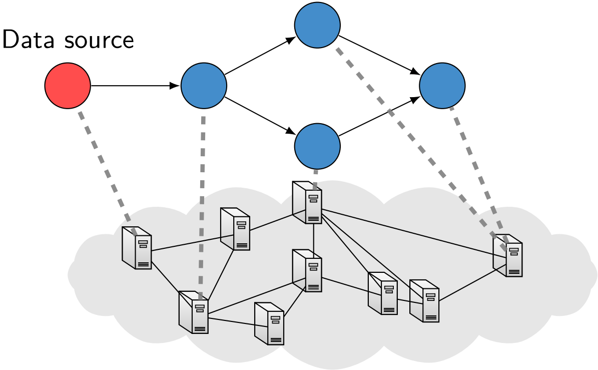
New trends for Big Data: geo-distributed processing



From large data centers in the Cloud to. . . **everywhere**

Executing DSP applications: placement

How to **place** the application components
over a computing infrastructure?



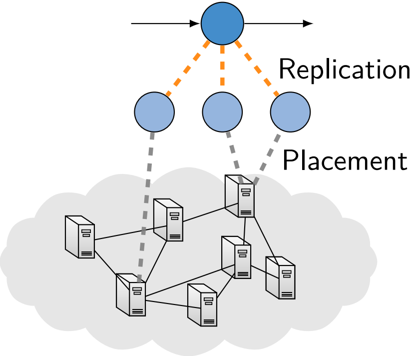
Network latency and resource heterogeneity impact the QoS!

A (centralized) optimization problem

EDRP

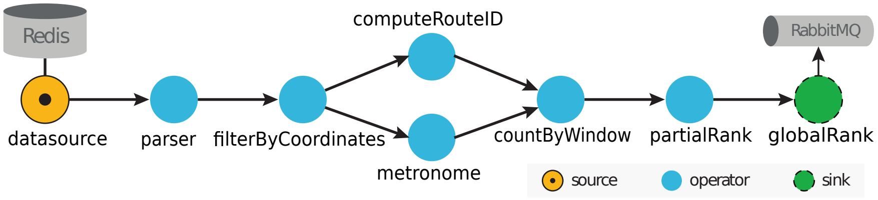
Elastic DSP Replication and Placement

- ILP model
- Optimizes trade-off between response time, resource usage, and reconfiguration cost
- Requires full characterization of the application and the infrastructure
- Does not scale!
- No foresight

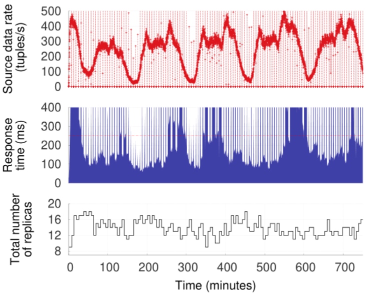


V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo, "Optimal operator deployment and replication for elastic distributed data stream processing", *Concurrency and Computation: Pract Exper.*, 2017

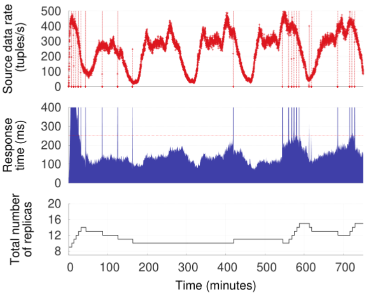
DEBS 2015 Grand Challenge application



EDF + Reinforcement learning



Q-learning



Q-learning, with token bucket