TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Addressing Deployment Challenges in Data Stream Processing

Corso di Sistemi e Architetture per Big Data

A.A. 2022/23

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

DSP deployment challenges

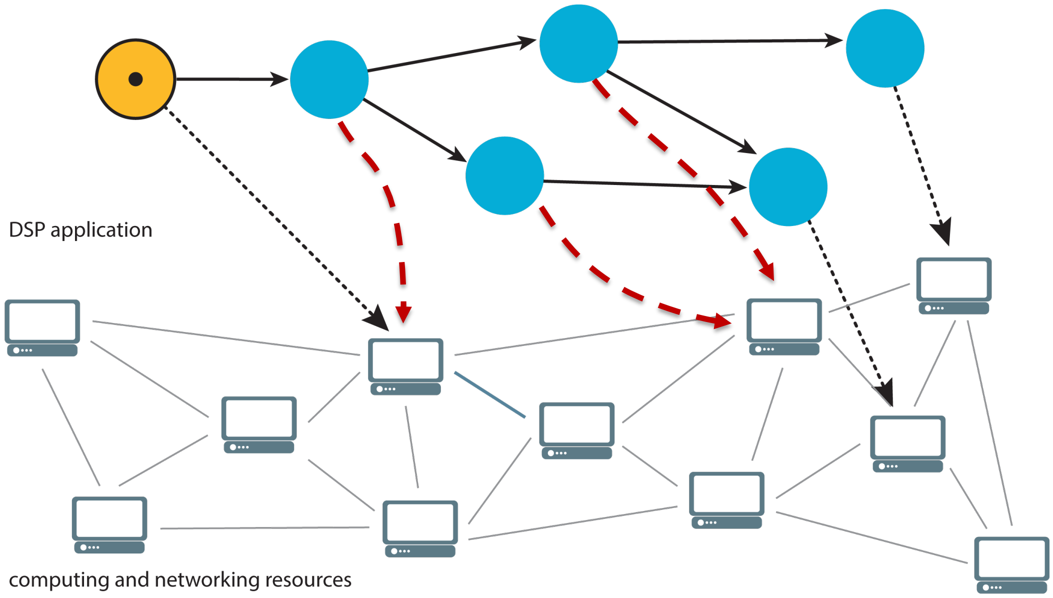
- Let's consider two challenges when deploying DSP applications

a) How to place DSP operators on underlying computing infrastructure (i.e., operator placement)

b) How to determine and adapt at run-time the number of replicas per operator (i.e., operator elasticity)

DSP operator placement

- Goal: determine which distributed computing nodes should host and execute each application operator, with the goal of optimizing application QoS



Placement: Edge-Cloud continuum

- **Edge/Fog + Cloud computing**: allows to increase scalability and availability, reduce latency, network traffic, and power consumption
- But placement becomes more challenging

Placement: challenges

- Significant network latencies
  - E.g., geo-distributed resources
- Heterogeneous computing and networking resources
  - E.g., capacity limits , business constraints
- Computing/network resources can be unavailable
- Data movement around the network

- Plus peculiarities of DSP applications:
  - Computational requirements may be unknown a-priori and change continuously
  - Long-running applications

→ Need to adapt to internal and external changes

Placement: frameworks

- Most frameworks use simple placement policies
- Apache Storm
  - Round Robin as default strategy
  - Resource Aware Scheduler as alternative

    storm.apache.org/releases/2.1.1/Resource_Aware_Scheduler_overview.html
    - Takes into account resource availability on machines and resource requirements of workloads
    - But requires user to specify memory and CPU requirements for individual topology components

Placement: different approaches

- Several operator placement policies in literature that address the problem but:
  - Different assumptions (system model, application topology, QoS attributes and metrics, …)
  - Different objectives
  - Not easily comparable
- Main methodologies:
  - **Mathematical programming**
    - Optimal operator placement problem: NP-hard
    - Does not scale well, but provides useful insights
  - **Heuristics**
    - Majority of policies
  - Deep Reinforcement Learning

Placement: different approaches

- Who is the decision maker?
  - **Centralized** placement strategies
    - Require global view (full resource and network state, application state, workload information)
    - ✓ Capable of determining optimal global solution
    - ✗ Scalability
  - **Decentralized** placement strategies
    - Take decision based only on local information
    - ✓ Scalability, better suited for run-time adaptation
    - ✗ Optimality is not guaranteed

ODP: Optimal DSP Placement

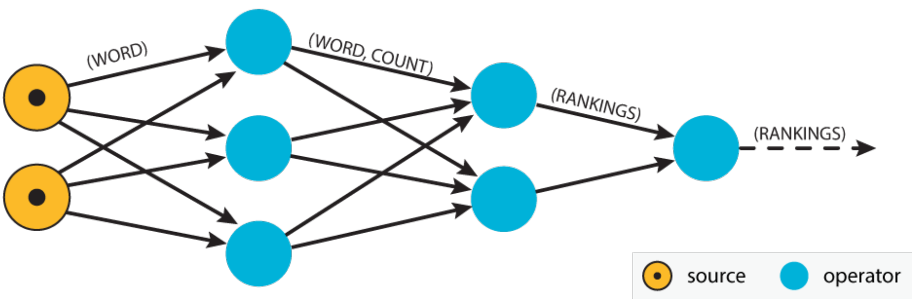
- We proposed ODP
  - Centralized policy for optimal placement of DSP applications
  - Formulated as Integer Linear Programming (ILP) problem

- Our goals:
  - To compute the **optimal placement** (of course!)
  - To provide a **unified general formulation** of the placement problem for DSP applications (but not only!)
  - To consider multiple **QoS attributes** of applications and resources
  - To provide a **benchmark** for heuristics

V. Cardellini, V. Grassi, F. Lo Presti, M. Nardelli, Optimal Operator Placement for Distributed Stream Processing Applications, DEBS '16

ODP: model

DSP application



Operators

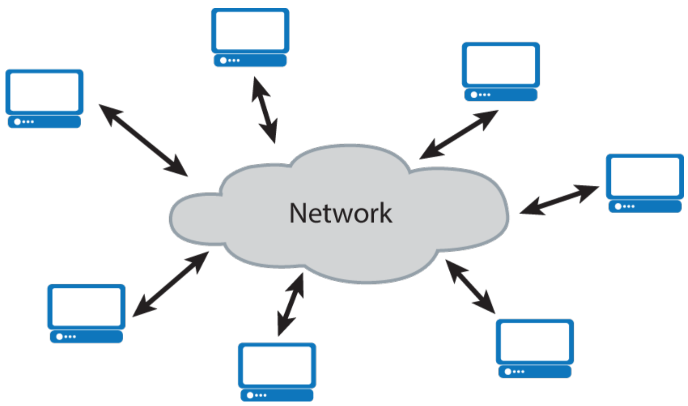
- C_i: required computing resources
- R_i: execution time per data unit

Data streams

- $\lambda_{i,j}$: data rate from operator *i* to *j*

ODP: model

Computing and network resources



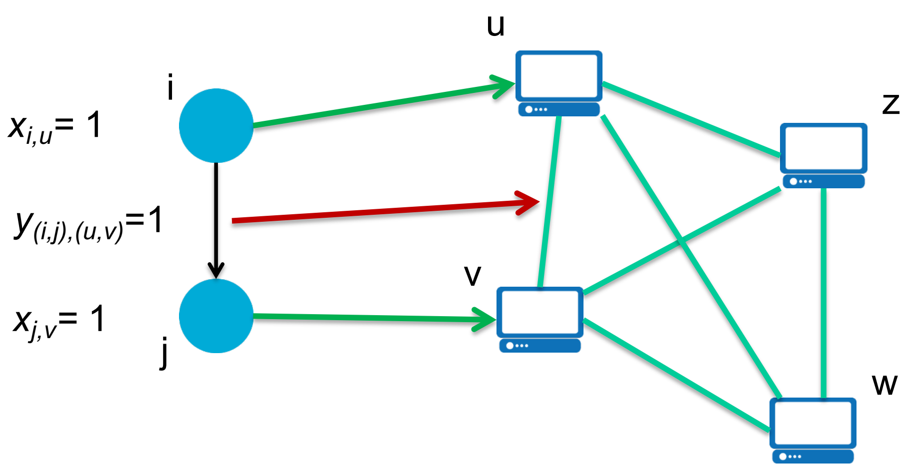
Computing resources
- C_u: amount of resources
- S_u: processing speed
- A_u: resource availability

(Logical) Network links
- $d_{u,v}$: network delay from u to v
- $B_{u,v}$: bandwidth from u to v
- $A_{u,v}$: link availability

ODP: model

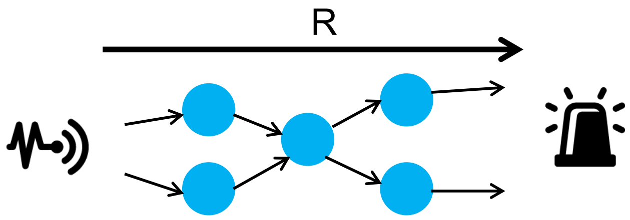
Decision variables

- Determine where to map DSP operators and data streams



ODP: some QoS metrics

- **Response time**
  max end-to-end delay between sources and destination



- **Application availability**
  probability that all components/links are up and running
- **Inter-node traffic**
  overall network data rate
- **Network usage**
  in-flight bytes

$\Sigma_{\text{links} \in l}$ rate(l)Lat(l)

ODP: optimal problem formulation

Tunable knobs to set the optimal placement goals

$$\boxed{\max_{\boldsymbol{x},\boldsymbol{y},r} F(\boldsymbol{x},\boldsymbol{y},r)}$$

subject to:

Latency

$$r \geq \sum_i \sum_u \frac{R_i}{S_u} x_{i,u} + \sum_{(i,j)} \sum_{(u,v)} d_{(u,v)} y_{(i,j),(u,v)} \qquad \forall p \in \pi_G$$

Availability

$$a(\boldsymbol{x},\boldsymbol{y}) = \sum_i \sum_u a_u x_{i,u} + \sum_{(i,j)} \sum_{(u,v)} a_{(u,v)} y_{(i,j),(u,v)}$$

Network bandwidth and node capacity constraints

$$B_{(u,v)} \geq \sum_{(i,j)} \lambda_{(i,j)} y_{(i,j),(u,v)} \qquad \forall u \in V_{res}, v \in V_{res}$$

$$\sum_i C_i x_{i,u} \leq C_u \qquad \forall u \in V_{res}$$

Assignment and integer constraints

$$\sum_u x_{i,u} = 1 \qquad \forall i \in V_{dsp}$$

$$x_{i,u} = \sum_v y_{(i,j),(u,v)} \qquad \forall (i,j) \in E_{dsp}, u \in V_{res}$$

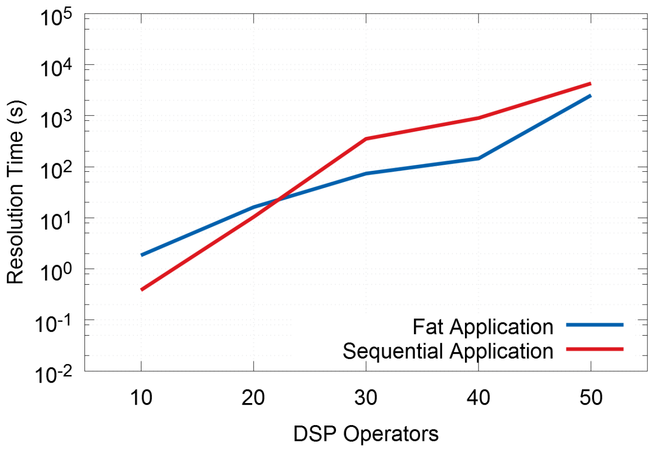
$$x_{j,v} = \sum_u y_{(i,j),(u,v)} \qquad \forall (i,j) \in E_{dsp}, v \in V_{res}$$

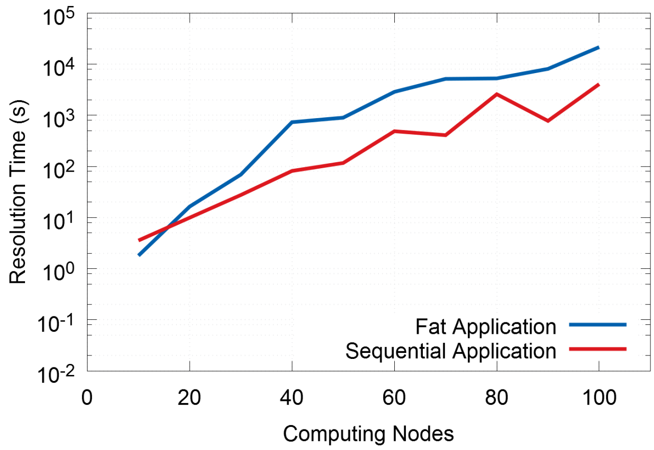
$$x_{i,u} \in \{0,1\} \qquad \forall i \in V_{dsp}, u \in V_{res}$$

$$y_{(i,j),(u,v)} \in \{0,1\} \qquad \forall (i,j) \in E_{dsp}, (u,v) \in E_{res}$$

ODP: scalability issue

Placement problem is NP-hard: does not scale well!





We need **heuristics** to compute placement
in a feasible amount of time

Centralized placement heuristics

- Example of centralized heuristic that aims to reduce inter-node traffic
- **Aniello et al.**: co-locate pairs of communicating tasks on same computing node as to minimize inter-node communication and balance CPU demand

  Greedy heuristic – Key idea:
  - Rank task pairs according to exchanged traffic
  - For each pair:
    - » If task pairs have not been yet assigned, assign them to same node
    - » If either is assigned, consider least loaded node and those where they have been assigned. Work out the configuration which minimizes the inter-process traffic

L. Aniello, R. Baldoni and L. Querzoni, Adaptive online scheduling in Storm, DEBS '13

Decentralized placement heuristic

- Heuristics goal: reduce network usage
  - Network usage metric combines link latencies and exchanged data rates among DSP operators:

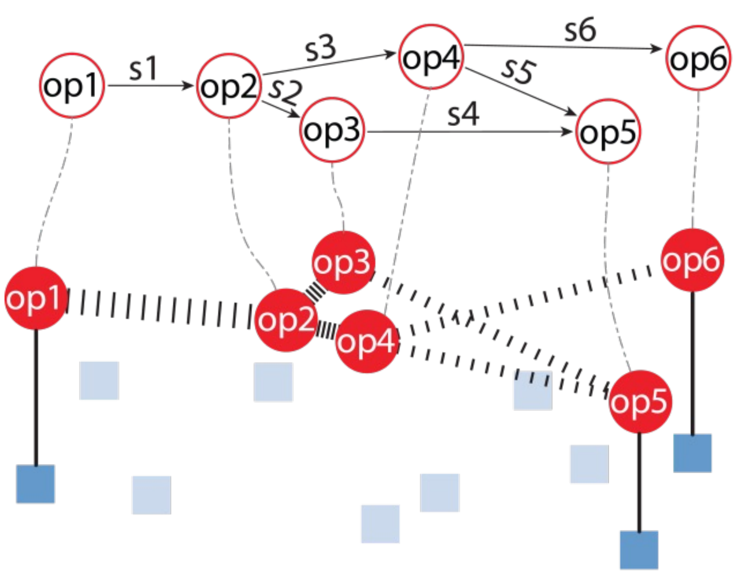
$$\sum_{links \in l} rate(l)Lat(l)$$

- **Pietzuch et al.** exploit **spring relaxation** idea:
  - DSP application regarded as a system of springs, whose minimum energy configuration corresponds to minimizing network usage
- Features
  - Decentralized policy to minimize network impact
  - Adaptive to change in network conditions

P. Pietzuch et al., Network-aware operator placement for stream-processing systems, ICDE '06

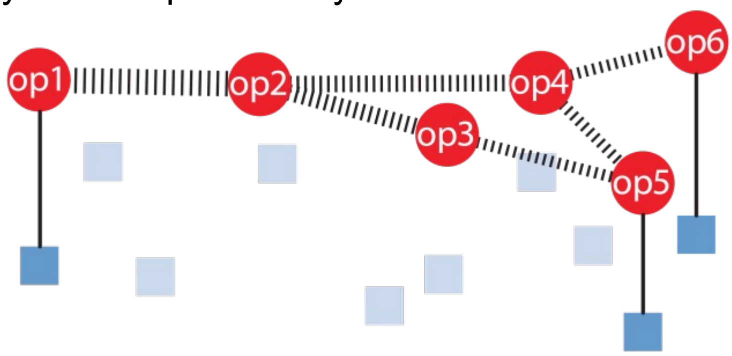
Decentralized placement heuristic

1. Represents DSP application as an equivalent system of springs



Decentralized placement heuristic

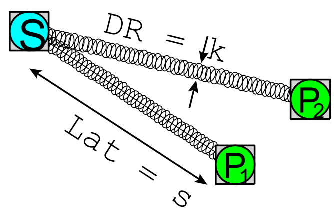
2. Determines operator placement in the cost space by minimizing the elastic energy of the equivalent system



Network of springs tries to minimize potential energy E

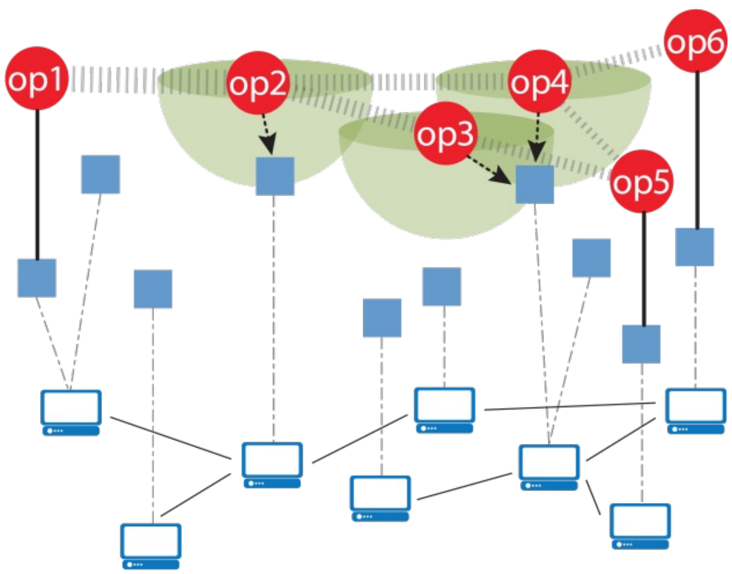
$$E = \sum_{l \in L} DR(l) Lat(l)^2$$

Streams as springs, that restore a force F = ½ • k • s:
- k (spring constant): exchanged data rate on link
- s (spring extension): latency on link



Decentralized placement heuristic

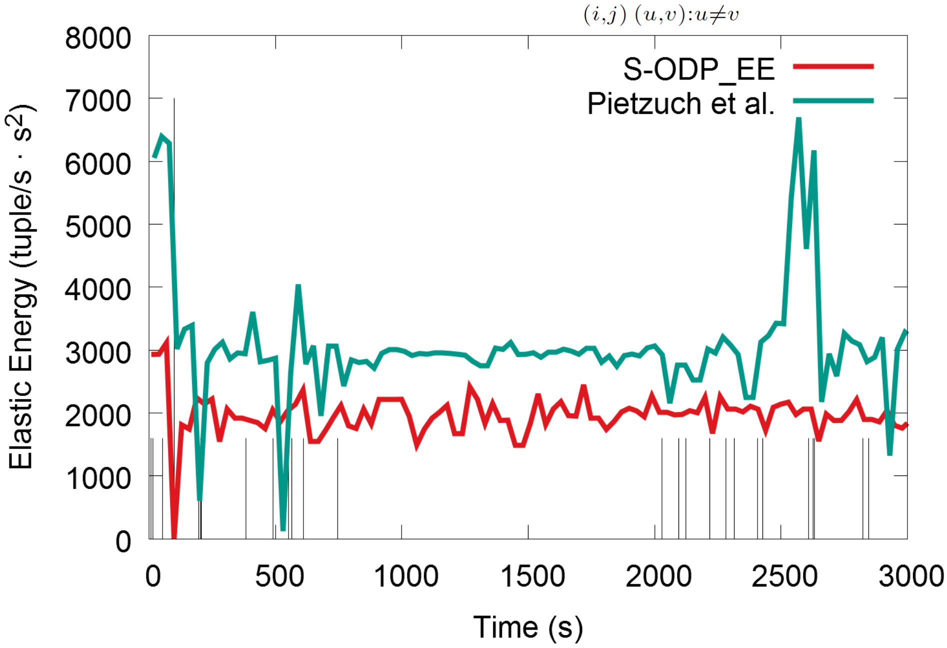
3. Maps its decision back to physical nodes



ODP as benchmark

Distributed placement heuristic that minimizes network usage

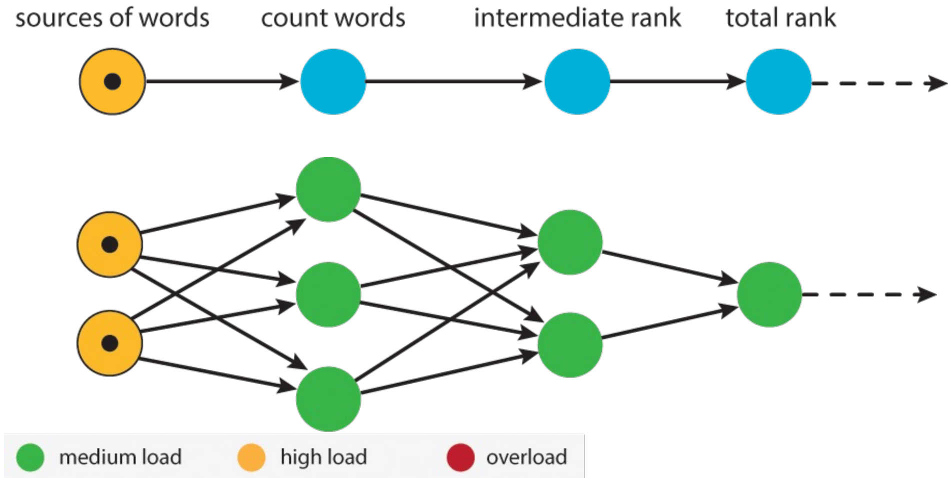
Pietzuch et al. : $\min EE(\boldsymbol{y}) = \min \sum_{(i,j)} \sum_{(u,v):u \neq v} \lambda_{(i,j)} d^2_{(u,v)} y_{(i,j),(u,v)}$



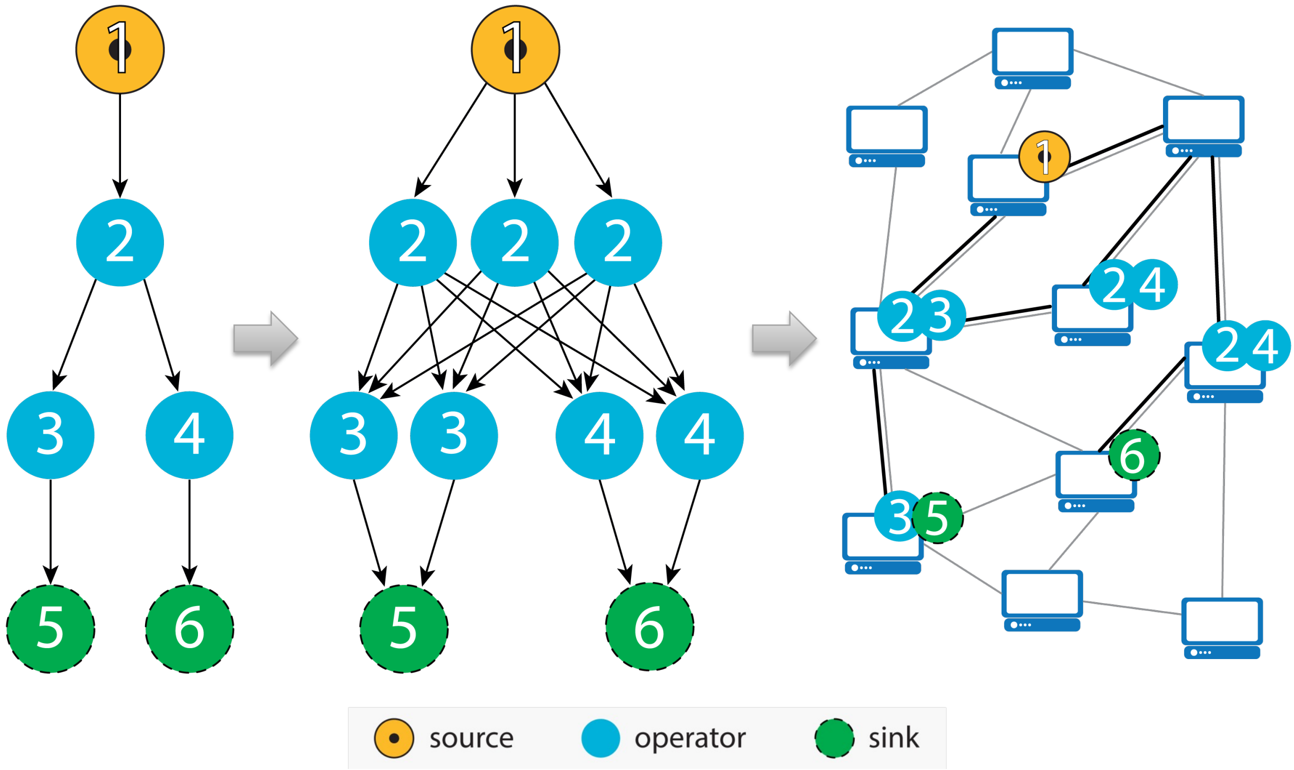
V. Cardellini, V. Grassi, F. Lo Presti, M. Nardelli, Optimal Operator Placement for Distributed Stream Processing Applications, DEBS '16

Not only placement

- Stream processing workloads are characterized by:
  - High volume and production rate
- Exploit replication (i.e., operator elasticity): concurrent execution of multiple operator replicas on different data portions
- How to determine the number of replicas?



Operator placement and replication



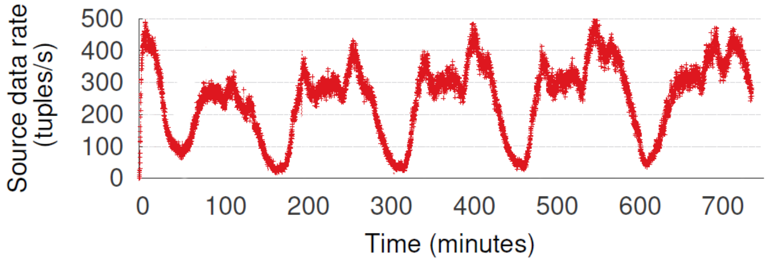
ODRP: Opt. DSP Replication and Placement

- We proposed **ODRP**
  - Centralized policy for **optimal replication and placement** of DSP applications
  - Formulated as Integer Linear Programming (ILP) problem that extends ODP
- Our goals:
  - Jointly determine optimal **number of replicas** and their **placement**
  - Consider multiple QoS attributes of applications and resources
  - Provide a unified general formulation
  - Provide a benchmark for heuristics
- Limitation: scalability, in practice we need heuristics

V. Cardellini, V. Grassi, F. Lo Presti, M. Nardelli, Optimal operator replication and placement for distributed stream processing systems, *ACM Perf. Eval. Rew.*, 2017.

DSP deployment challenges

- How to self-adapt at run-time the application deployment?
- DSP applications are:
  - long-running
  - subject to varying workloads
  - with



- Which main mechanisms do we need for run-time adaptation?
  - **Migration**: move operators from one node to another
  - **Elastic scaling**: change parallelism at application and/or infrastructure level

Elasticity: limits of centralized approaches

- Centralized optimization algorithms do not scale for large problem instances
- Centralized MAPE architecture does not scale in geo-distributed environments
  - Components are distributed but control logic is still centralized
- Which solution for Edge-Cloud continuum? **Decentralize MAPE**

How to decentralize control?

- Many patterns for decentralized control
  – Each one having pros and cons

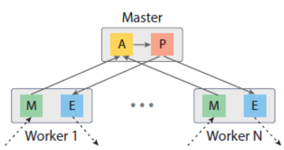


Figure 1: Hierarchical MAPE: master-worker pattern

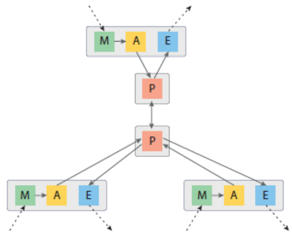


Figure 2: Hierarchical MAPE: regional pattern

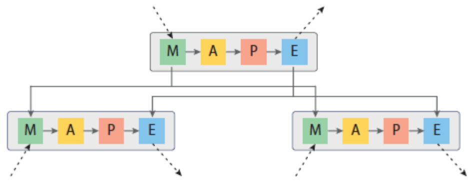


Figure 3: Hierarchical MAPE: hierarchical control pattern

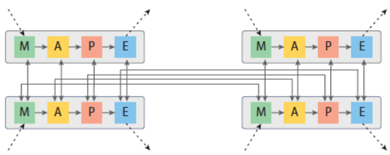


Figure 4: Flat MAPEs: coordinated control pattern

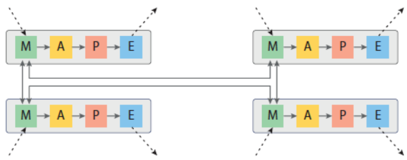
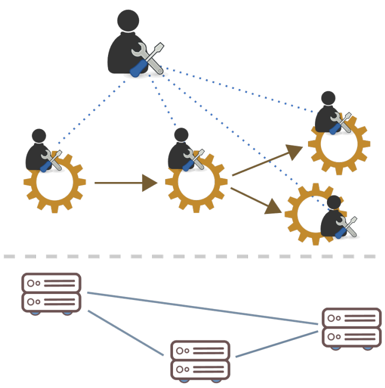


Figure 5: Flat MAPEs: information sharing pattern

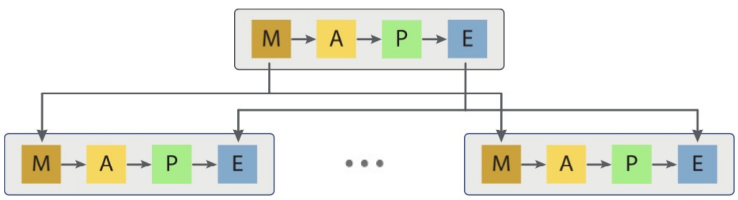
D. Weyns et al., On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, 2013

How to decentralize control?

- Our approach:
  – **Hierarchical distributed** architecture to support run-time adaptation
  – Based on efficient distribution of **MAPE** control loops



Global view



Local views

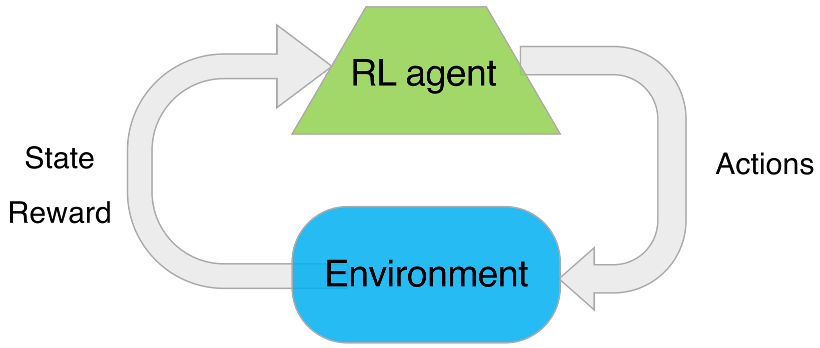
Local elasticity policy

- Let's focus on the local policy to control the elasticity of each DSP operator
- The policy can rely only on limited local view of system
  – e.g., utilization and input data rate of the operator it controls
- Two classes of elasticity policies
  – Classic threshold-based policy (e.g., used by AWS Auto Scaling)
    ✗ Need experience to choose thresholds
  – Based on Reinforcement Learning

V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo, Decentralized self-adaptation for elastic Data Stream Processing, *Future Generation Computer Systems*, 2018

Reinforcement Learning in a nutshell

- A branch of ML dealing with sequential decision-making
- Agent interacts with environment through actions and receives feedback in the form of reward (paid cost)
- Goal: learn to act as to maximize (minimize) long-term reward (cost)
- Trial-and-error experience

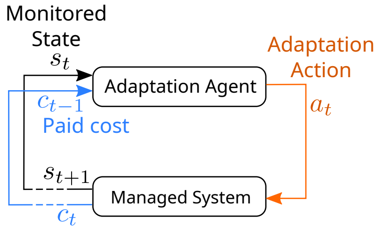


Reinforcement Learning in a nutshell

- We consider different classes of RL algorithms:
  - Baseline **model-free** learning algorithms (e.g., Q-learning)
  - **Model-based** learning algorithms that exploit what is known or can be estimated about system dynamics

Sutton and Barto, Reinforcement Learning: An Introduction, 2020

RL-based local elasticity policy



- At each step RL agent performs an action, looking at current state s_t
- Chosen action a_t causes payment of immediate cost c_t and transition to a new state s_{t+1}
- To minimize expected long-term (discounted) cost, RL agent estimates $Q(s, a)$
  - Q-function: expected long-run cost that follows the execution of action *a* in state *s*

Algorithm 1 RL-based Operator Elastic Control Algorithm

```
Initialize the Q functions
loop
    choose a scaling action a_i (based on current estimates of Q)
    observe the next state s_{i+1} and the incurred cost c_i
    update the Q(s_i, a_i) functions based on the experience
end loop
```

RL-based local elasticity policy: Q-learning

- Q-learning: baseline model-free RL algorithm
- Given current state, the agent chooses next action
  1. Either exploiting its knowledge about system (i.e., current estimates of Q-function stored in Q-table) by greedily selecting the action that minimizes the estimated future costs
  2. Or exploring by selecting a random action to improve its knowledge about system
     - We consider ε-greedy action selection method

Q-table

State/Action	a_1	a_2	…
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	…
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	…
…	…		
s_n	$Q(s_n, a_1)$	$Q(s_n, a_2)$	…

- Q-learning: update step of Q-function

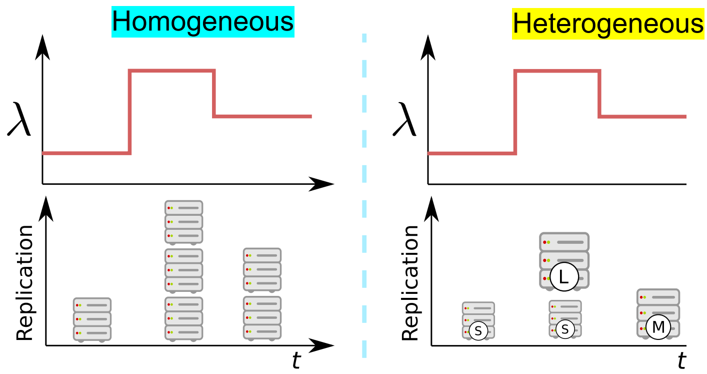
$$Q\left(s_i, a_i\right) \leftarrow (1-\alpha) Q\left(s_i, a_i\right)+\alpha\left[c_i+\gamma \min _{a^{\prime} \in \mathcal{A}\left(s_{i+1}\right)} Q\left(s_{i+1}, a^{\prime}\right)\right]$$

RL-based local elasticity policy: advanced RL techniques

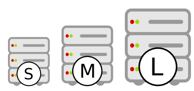
- We have exploited advanced RL techniques in order to deal with large state space (e.g., due to heterogeneous computing resources)
  – Function Approximation
  – Deep Learning
  – Goal: build approximate representations of state space and achieve near-optimal solutions with reduced memory demand
- Let’s consider the high-level ideas
- To learn more about:
  – Our tutorial at Performance 2021 Reinforcement Learning for Run Time Performance Management in the Cloud/Edge
  – Russo Russo et al., Hierarchical Auto-Scaling Policies for Data Stream Processing on Heterogeneous Resources, ACM TAAS, 2023

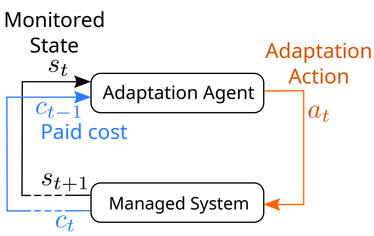
Auto-scaling on heterogeneous nodes

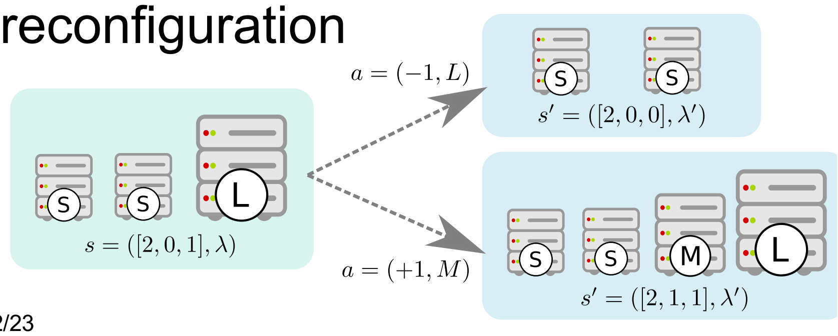
- We consider a heterogeneous computing infrastructure
  - Nodes with different types/amount of resources
- RL agent must decide not only how many replicas to run but also which types of nodes to host them



How to formulate?

- N resource types: T_{res} = {  }
- State s = (**k**, λ)
  - k_i = #replicas on nodes of type i
  - λ = input data rate
- Actions A(s)={(δ,τ): δ∈{−1,+1}, τ ∈T_{res}}∪{do−nothing}
- Cost = w_{res} resource cost + w_{perf} performance + w_{rcf} reconfiguration

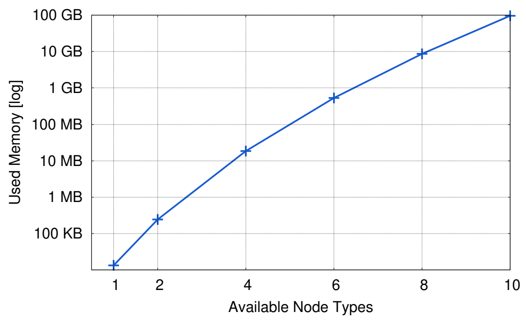


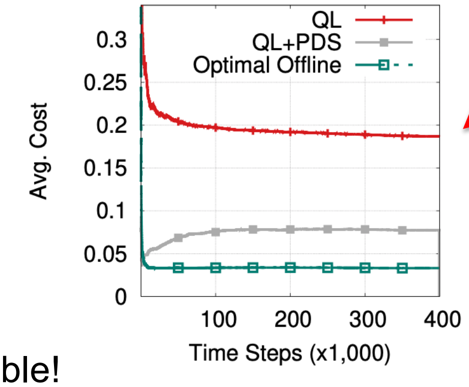


Standard RL algorithms do not work

- Q-learning does not work
  - ✗ Too much memory to store tabular representation of Q-function
  - ✗ Very slow convergence

State/Action	a_1	a_2	...
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	...
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	...
...		...	
s_n	$Q(s_n, a_1)$	$Q(s_n, a_2)$	...





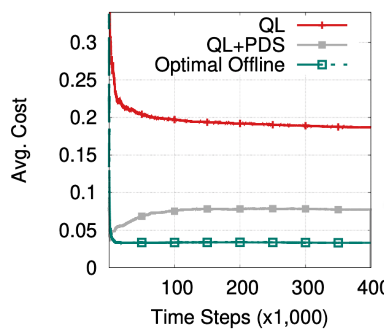
Q-learning

Note: each operator has its own Q-table!

How to improve?

- We exploit multiple solutions

1. Separate the known from the unknown, inject **partial model knowledge** (i.e., post-decision states) and learn only the unknown part
   - Do we really need to learn everything from scratch?
     - We know which is the impact of scaling actions on the current deployment
     - We know whether a reconfiguration cost is paid after a certain action
     - We can estimate performance-related costs through a model

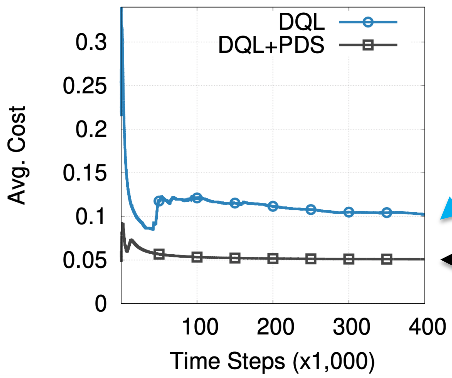


Partial model knowledge

How to improve?

- We exploit multiple solutions

2. Resort to non-linear **function approximation** (deep Q network)
3. Combine all together



Function approximation

Partial model knowledge
+
Function approximation

Other DSP deployment challenges

- DSP applications and serverless DSP in the Edge-Cloud continuum?
- How to provide security guarantees?
- **Thesis topics**