A Multi-level Elasticity Framework for Distributed Data Stream Processing

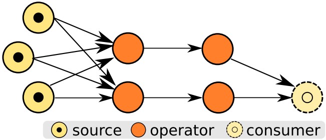
Matteo Nardelli, **Gabriele Russo Russo**,
Valeria Cardellini, Francesco Lo Presti

University of Rome Tor Vergata, Italy

Università di Roma
Tor Vergata

Distributed Data Stream Processing (DSP)

- Data streams continuously generated by distributed sources (e.g., IoT sensors)
- Near real-time processing



More and more strict processing latency requirements

↓

Need to push computation from the Cloud towards data sources and consumers
(**Fog Computing**)

DSP & Fog: old and new challenges

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

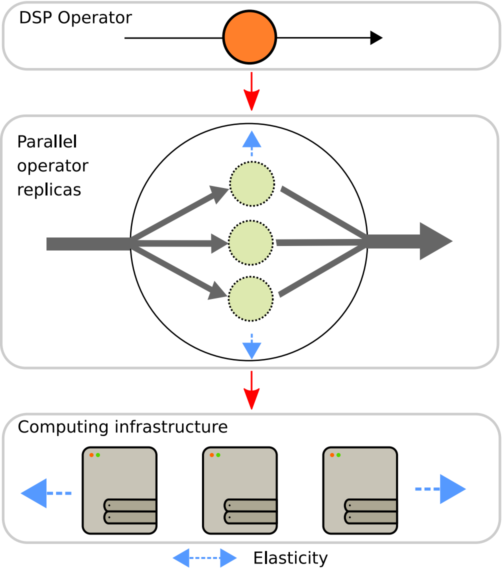
↓

- Application deployment critical for Quality of Service
- Long-running nature of DSP apps calls for run-time adaptation
- Large distributed infrastructures cannot be managed by hand

Multi-level Elasticity

Application-level elasticity
Adjusting the operators parallelism in response to workload variations

Infrastructure-level elasticity
Provisioning computing resources as needed to reduce operating costs and energy consumption



State of the art

Infrastructure-level elasticity

- widely investigated for VM auto-scaling in the Cloud
- a few solutions for Fog Computing scenarios

Application-level elasticity for DSP

- many different policies (thresholds, queuing theory, ML, ...)
- EDF, Elastic Distributed DSP Framework:
  hierarchical decentralized elasticity

> V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo,
> "Decentralized self-adaptation for elastic data stream processing",
> *Future Generation Computing Systems*, 2018.

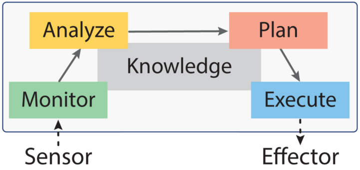
Multi-level elasticity for DSP: only centralized solutions so far

Goals

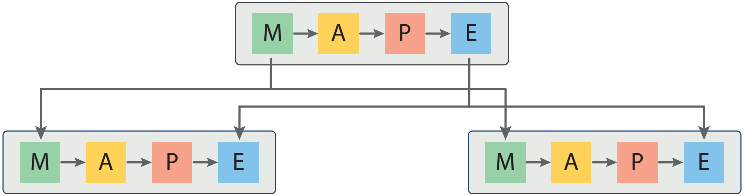
- Designing a framework for the autonomous control of geo-distributed DSP
- Supporting both application-level and infrastructure-level elasticity
- Defining a set of simple elasticity control policies
- Integrating the framework in Apache Storm

Hierarchical Self-Adaptation

MAPE: **M**onitor **A**nalyze **Pl**an **E**xecute
reference pattern for self-adapting systems

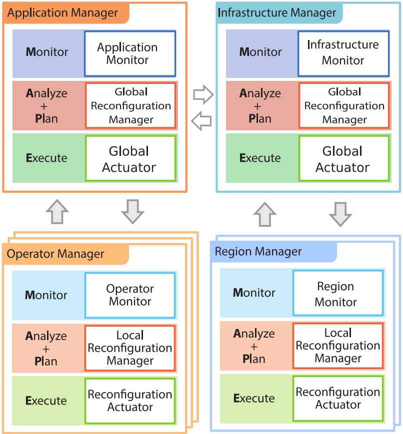


Hierarchical MAPE pattern
decentralized self-adaptation
with separation of concerns and time scales



E2DF: 2-level Elasticity Framework

Application Control System + Infrastructure Control System, each designed according to the hierarchical MAPE pattern



The **ACS** is responsible for the application deployment

The **ICS** manages the computing infrastructure, composed of regions

Must cooperate!

Application Control System

- Each **Operator Manager** (OM) monitors a single DSP operator
- OMs plan reconfigurations for an operator based on a local policy, and propose them to the Application Manager

- The **Application Manager** (AM) supervises a whole DSP application, aiming at meeting some QoS requirements
- Each AM collects requests from the OMs, and grants/rejects them based on its global policy

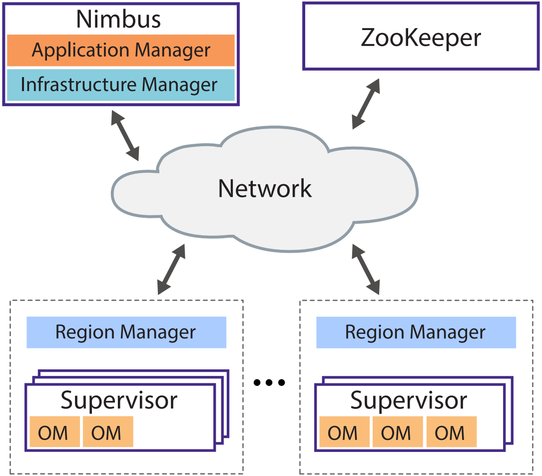
Infrastructure Control System

- **Region Managers** (RM) responsible for resource allocation (VM, containers, ...) in each region
- RMs issue reconfiguration requests to the IM based on a local policy

- The **Infrastructure Manager** (IM) supervises the whole infrastructure
- Collects requests from all the regions, and grants/rejects them based on its global policy
- Interacts with one or more Application Managers when necessary

Integration in Apache Storm

We build on top of Distributed Storm:
stateful migration, extended QoS monitoring, . . .



Simple Control Policies: ACS

Operator Manager:

- proposes to scale-out,
  when average replica CPU utilization is larger than $\bar{U}$
- proposes to scale-in,
  when utilization with less replicas would be less than $\bar{U}$

Application Manager:

- rejects reconfigurations trying to acquire
  the same computing resource
- accepts all the others

Simple Control Policies: ICS

Region Manager:

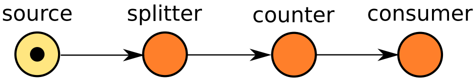
- C_r, minimum amount of available "slots" in each region r
- proposes to launch new instances
  when available capacity is less than C_r
- proposes to kill unused instances in case of over-provisioning
- proposes to kill used nodes with very low utilization
  (after migrations!)

Infrastructure Manager:

- grants all reconfiguration requests
- interacts with Application Managers when a node could be turned off after migrating the operator replicas

Evaluation

- *WordCount* topology



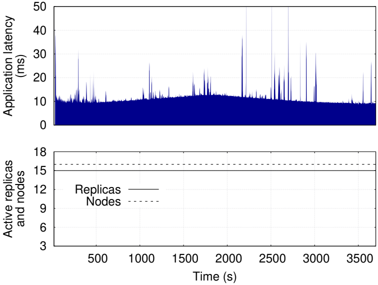
- Simple increasing and decreasing workload (5-550 tuple/s)
- Storm worker nodes instantiated as Docker containers

Three scenarios:

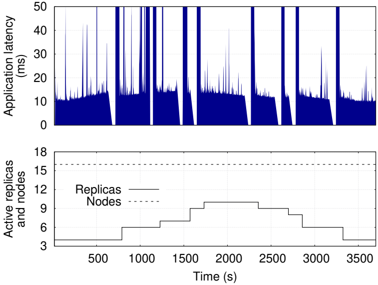
- No run-time adaptation
- Application-level elasticity only
- Application- and Infrastructure-level elasticity (E2DF)

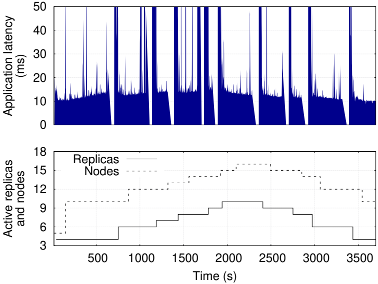
Results

Baseline (no adaptation)



ACS only





← E2DF

	Latency	**Nodes**	**Replicas**
Base	11 ms	16	15
ACS	19 ms	16	**6.2**
E2DF	19 ms	**12.5**	**6.1**

What's next?

We are investigating more complex policies
➔ e.g. **Reinforcement Learning**

System state: $s = (k, u, f)$
k ➔ number of active nodes
u ➔ avg. hosted replicas utilization
f ➔ (boolean) presence of any unused node

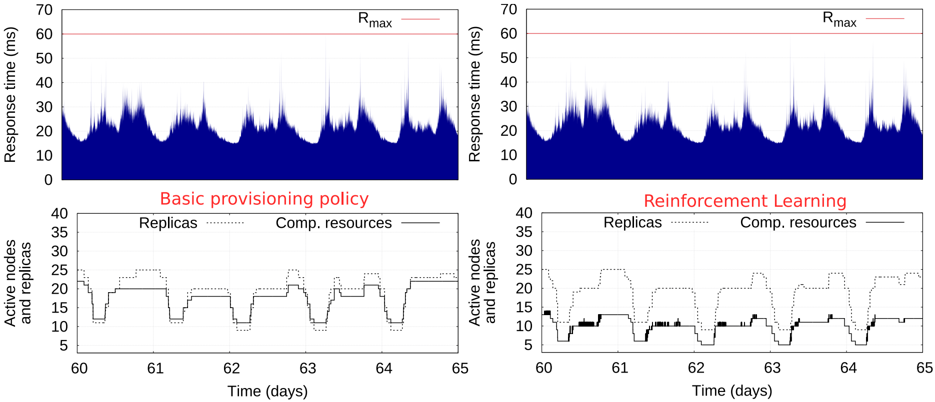
Actions: $\{-1, 0, +1\}$

Cost associated to state-action pair (s, a):

$$c(s, a) = c_{demand}(s, a) + c_{resources}(s, a)$$

Goal: minimizing the long-term cost!

E2DF with RL: preliminary results



Conclusions

- E2DF, a framework for hierarchical autonomous control of DSP application and resource elasticity
- Integrated in Apache Storm
- Simple yet effective control policies

Future work:

- More complex control policies (e.g., Reinforcement Learning)
- Vertical elasticity
- Implementation on top of other DSP frameworks

Thanks for your attention!

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