A Fully Decentralized Autoscaling Algorithm for Stream Processing Applications

Mehdi Belkhiria
Supervised by: Cédric Tedeschi

Univ Rennes, Inria, CNRS, IRISA, Rennes, France

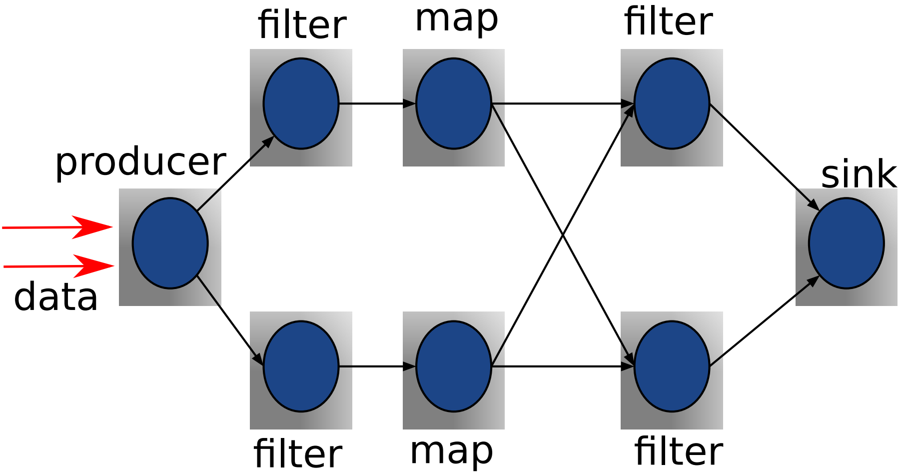
Auto-DaSP 2019 - Göttingen - August 27, 2019

Stream Processing

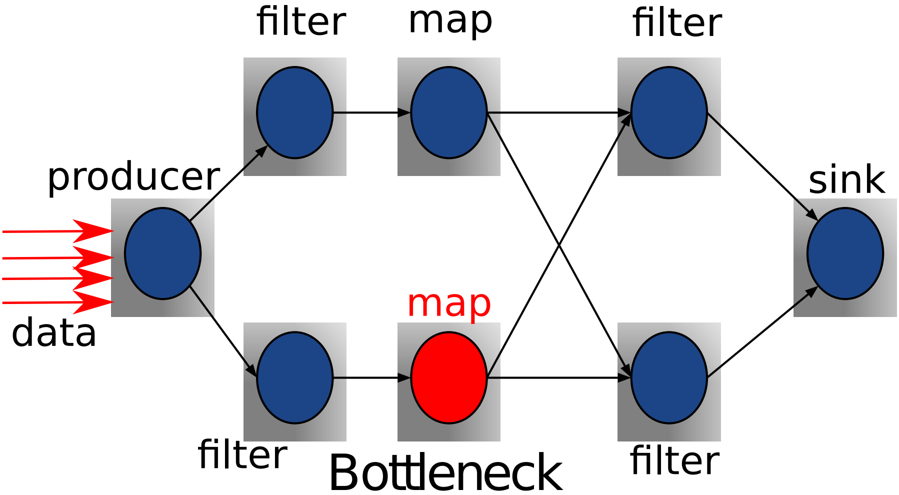
Stream processing is the process of being able to quickly produce some results in **real time**.

- The stock market
- Security (marine, aerial, ...)
- Bank transactions
- Weather forecasts

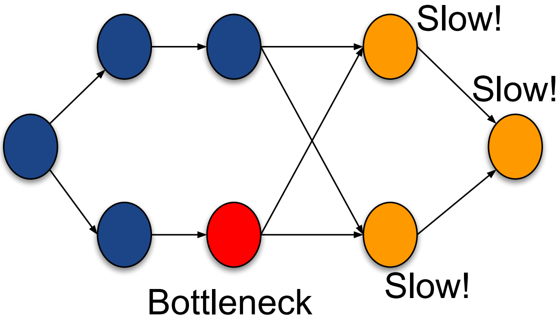
Context: Autoscaling In Stream Processing



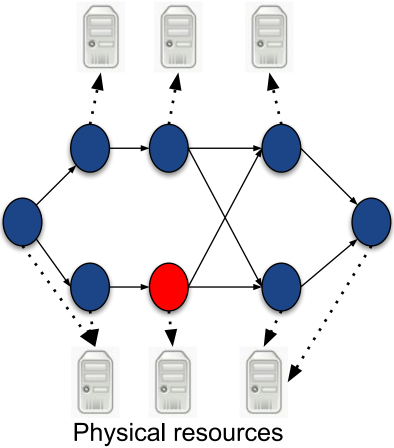
Context: Autoscaling In Stream Processing



Context: Autoscaling In Stream Processing

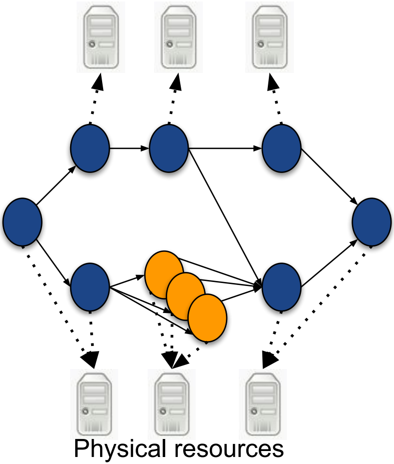


Context: Autoscaling In Stream Processing



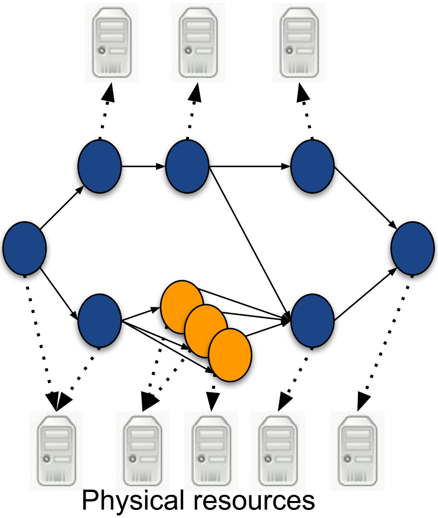
Maintain dynamically the right number of replicas for each operator

Context: Autoscaling In Stream Processing

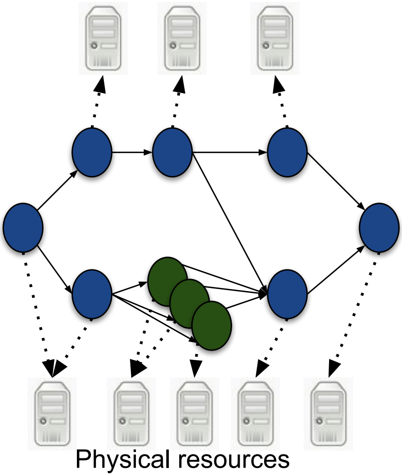


Fission method

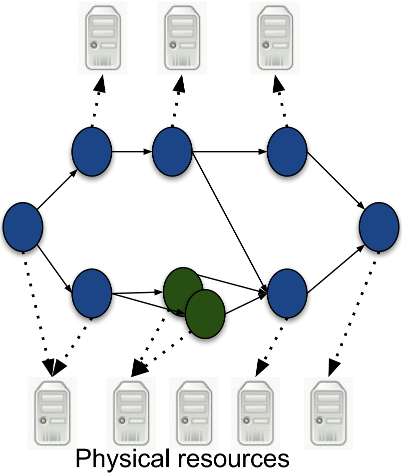
Context: Autoscaling In Stream Processing



Context: Autoscaling In Stream Processing

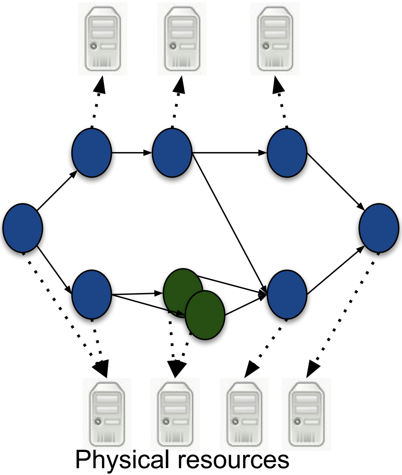


Context: Autoscaling In Stream Processing



Deletion method

Context: Autoscaling In Stream Processing



The elasticity should be **autonomous** and **without interruption**.

State Of The Art

State of the art of AutoScaling in SP

- Static approach [Scott Schneider et al. 2012]
- Dynamic approach
  - Centralised [Bugra Gedik et al. 2014]
  - Hierarchical [Cardellini et al. 2018]
  - Decentralised [Nicolò M. et al. 2012]

Our contribution

Fully decentralised

- Local scaling decision
- Maintain the view of the neighbours in concurrent settings

Table of Contents

System Model

Infrastructure

- Unbounded set of *reliable homogeneous* computation nodes: either physical or VMs
- Reliable FIFO channels

Application

- Stream processing applications (directed pipeline)
- Operators are *stateless*
- Each node hosts one operator
- Each operator knows only its successors and predecessors in the graph

SASO Properties

1. **S**tability
   $\rightarrow$ does not oscillate number of instances
2. **A**ccuracy
   $\rightarrow$ finds the number of instances that maximizes the throughput
3. **S**ettling time
   $\rightarrow$ reaches a stable number of instances quickly
4. **O**vershoot
   $\rightarrow$ does not use more instances than necessary

Local Decision

- Each node has its own elastic manager
  $\rightarrow$ taking decision with local information
  $\rightarrow$ local graph's knowledge should be the same in all sibling nodes

- These local decisions are set up through probabilistic duplications and deletions

Scaling Decision (1)

Local decision and with local information

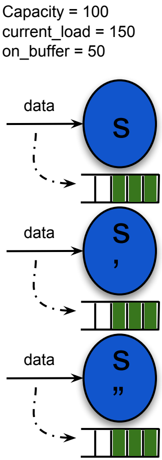
Constant parameters

- C: node's capacity
- L_{sh}: desired load's ratio

variable

- n_t: number of current instances
- l_t: current load

Scaling Decision (2)



Scaling Decision (2)

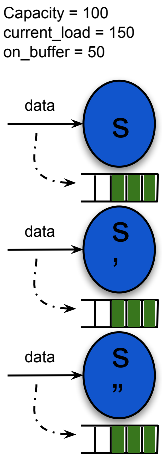
Algorithm 1 *getProbability*(C, r, l_t, n_t) : *float*

Require: C: Processing capacity
Require: r: Target load ratio
Require: l_t: current OI's load
Require: n_t: current number of siblings

$L_t \leftarrow l_t \times n_t$

$n_{diff} \leftarrow \left|\frac{L_t}{r \times C} - n_t\right|$

Return $\frac{n_{diff}}{n_t}$



Scaling Decision (2)

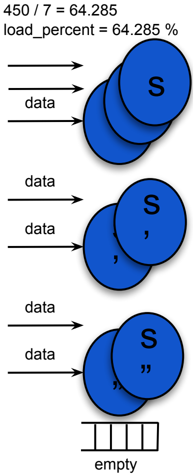
Algorithm 2 *getProbability*(C, r, l_t, n_t) : *float*

Require: C: Processing capacity
Require: r: Target load ratio
Require: l_t: current OI's load
Require: n_t: current number of siblings

$L_t \leftarrow l_t \times n_t$

$n_{diff} \leftarrow \left|\frac{L_t}{r \times C} - n_t\right|$

Return $\frac{n_{diff}}{n_t}$



Protocol

Objectif

Maintain consistent views of neighbours as concurrent scaling actions are triggered so no data message is lost

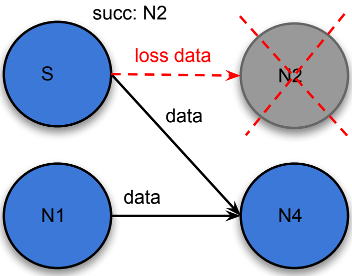
Three phases in the duplication case and two phases in deletion case

- send a scaling message
- send back an *acknowledge* messages
- send an activation message (in duplication case)

Concurrency Issues

What if we had concurrent duplication and/or deletion?

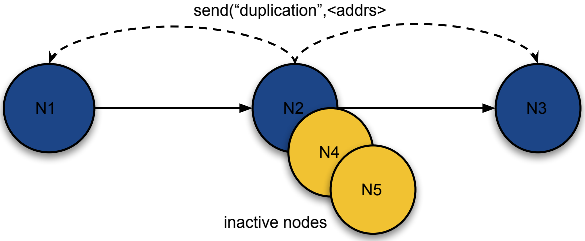
- A destroyed or new node could not be recognized by their predecessors
  → data loss
  → inactive nodes in the cluster



Protocol Steps

Step 1

- send a duplication message



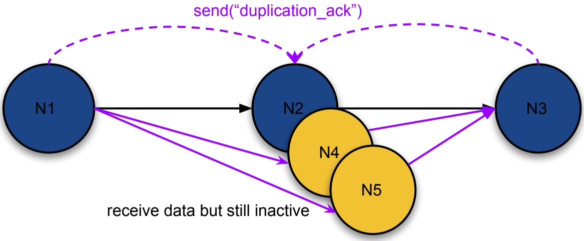
On Receipt

- **upon** receipt **of** $(``duplication", < addrs >)$ from $N2$

Protocol Steps

Step 2

- send back an acknowledge messages



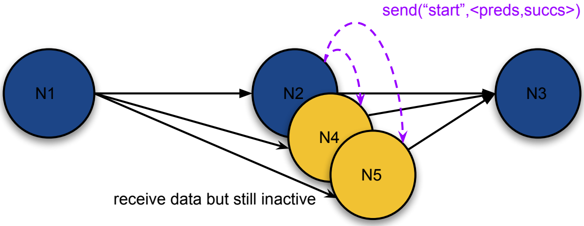
On Receipt

- **upon** receipt **of** (“*duplication_ack*”) **from** $N1$ and $N3$

Protocol Steps

Step 3

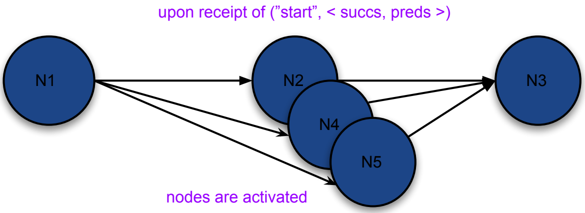
- send an activation message



On Receipt

- **upon** receipt **of** (“*start*”, $< succs, preds >$) **from** $N2$

Protocol Steps



- New nodes are active and can process data

Simulation Objectives & Setup

Simulation objectives

- Check message synchronization
- Estimate overhead messages sent in the network
- Estimate efficiency of the nodes' auto-scaling
- Compare the obtained throughput with the ideal throughput

Simulation setup

- Capacity $C = 500$
- Ideal load ratio $r = 0.7$
- top_threshold $thres_{\downarrow} = 0.8$
- down_threshold $thres_{\uparrow} = 0.6$
- Nodes try to start the scaling protocol every 5 steps
- Simulation runs for 200 steps.

Simulation Results

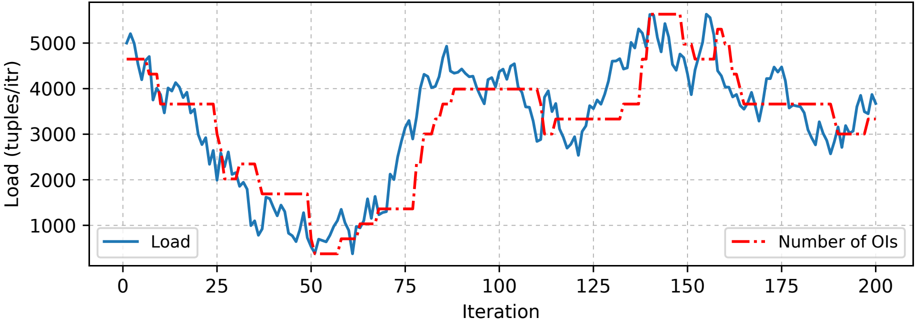
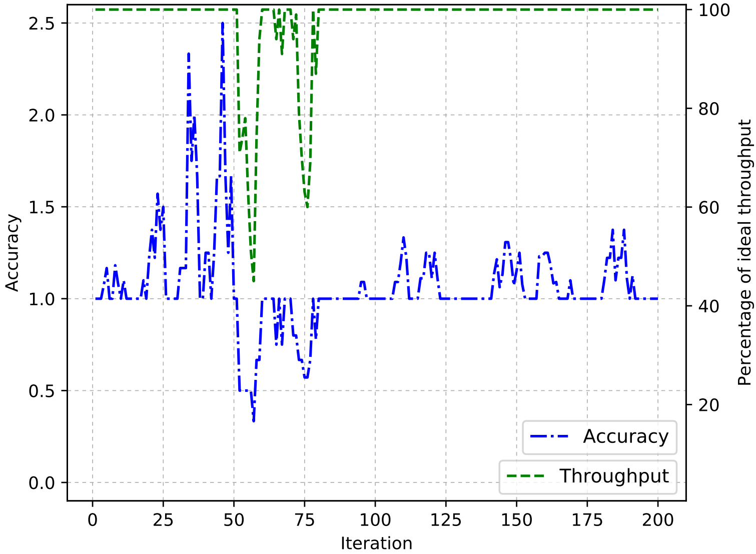


Figure 1: Load and number of nodes, local scale.

Simulation Results



Simulation Results

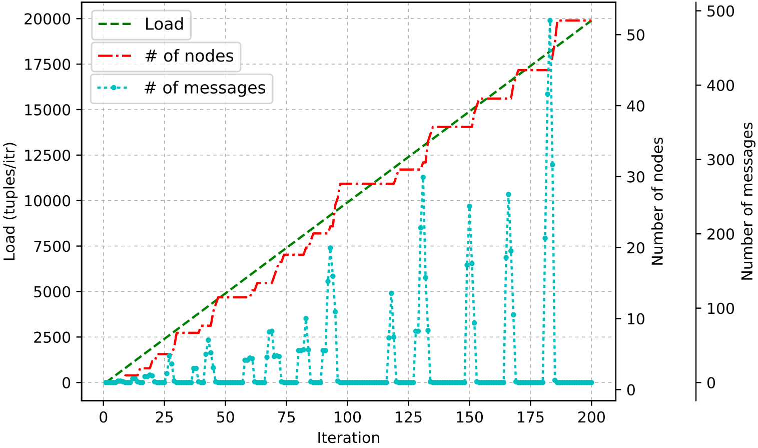


Figure 1: Load and number of nodes, local scale.

Current & Future Work

Experiments

- Development of a software prototype of a decentralized SPE
- Based on Kafka and Kafka Stream
- Currently being deployed over Grid'5000 (a nation-wide experimental testbed)

Algorithms extension

- Fault tolerance
- Heterogeneity
- Stateful operators