

Challenges in Data Stream Processing

Corso di Sistemi e Architetture per Big Data

A.A. 2023/24

Valeria Cardellini

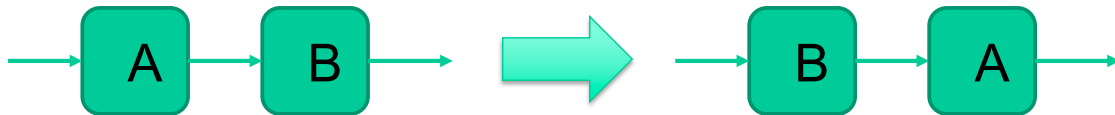
Laurea Magistrale in Ingegneria Informatica

Challenges

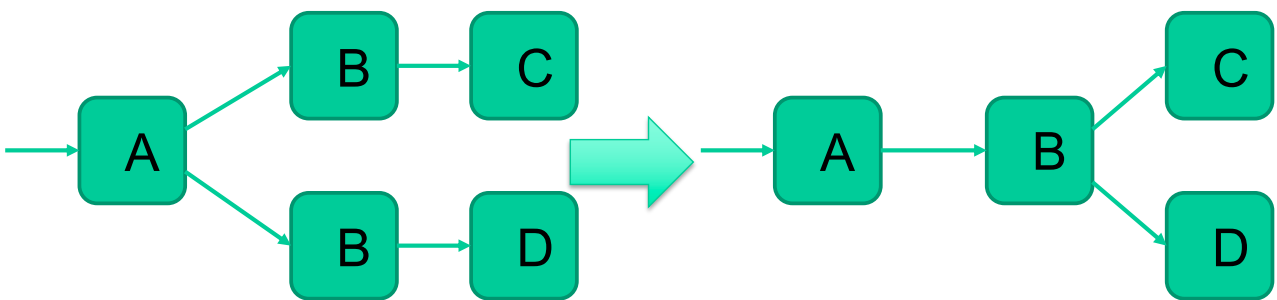
- Let's consider how to tackle some challenges in DSP systems
 1. Optimize DSP application
 2. Place DSP operators on computing infrastructure
 3. Manage load variations
 4. Self-adapt at run-time
 5. Manage stateful operators
 6. Fault tolerance

Challenge 1: Optimize DSP application

- Apply optimizations on the streaming graph
 - At design time or run-time
- Operator reordering
 - To avoid unnecessary data transfers



- Redundancy elimination

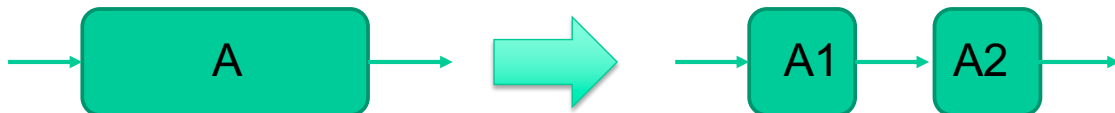


Valeria Cardellini - SABD 2023/24

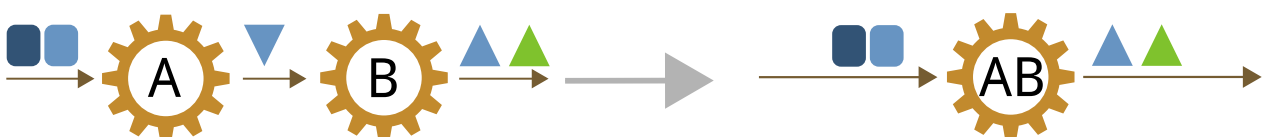
2

Challenge 1: Optimize DSP application

- Operator separation



- Operator fusion

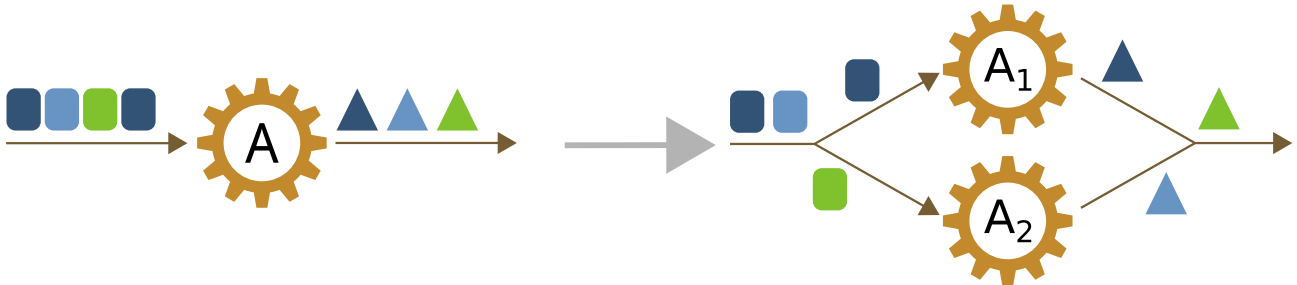


Valeria Cardellini - SABD 2023/24

3

Challenge 1: Optimize DSP application

- Operator scaling (aka *operator fission*)

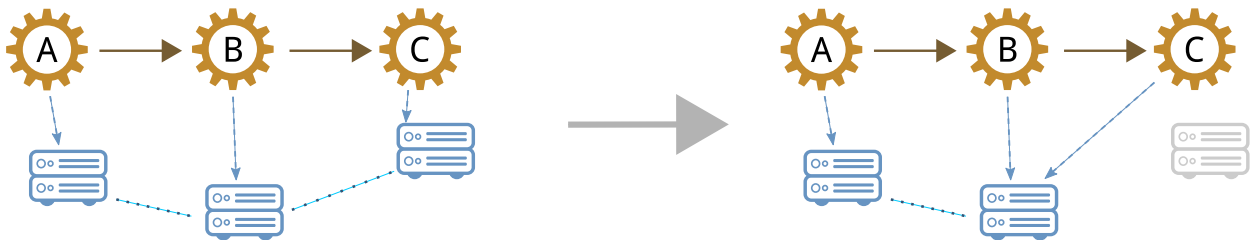


At the streaming system layer

- DSP application optimization is addressed at *DSP application layer* and usually at design time
- What about the *streaming system layer*?
- What about *run-time adaptation*?
- Let's first consider two solutions to improve performance (e.g., to control application latency) at the streaming system layer
 - **Place DSP operators**
 - **Manage load variations**

Challenge 2: Place DSP operators

- Determine, within a set of available distributed computing nodes, those nodes that should host and execute each operator instance of a DSP application

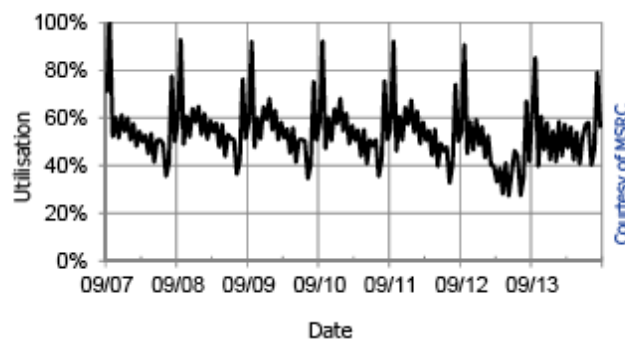


Challenge 2: Place DSP operators

- Operator placement decision: complex problem
 - Trade communication cost against resource utilization
- When
 - **Initial** (static) operator placement
 - Can be more expensive and comprehensive
 - Can also be **at run-time**
 - Place again all the operators or only a subset
 - Require **self-adaptation**
- We will focus on this issue later

Challenge 3: Manage load variations

- Typical stream processing workloads are:
 - with high volume and high rates
 - bursty and with workload spikes unknown in advance and difficult to predict
 - Twitter in 2013: rate of tweets per second = 5700
 - ... but significant peak of 144,000 tweets per second

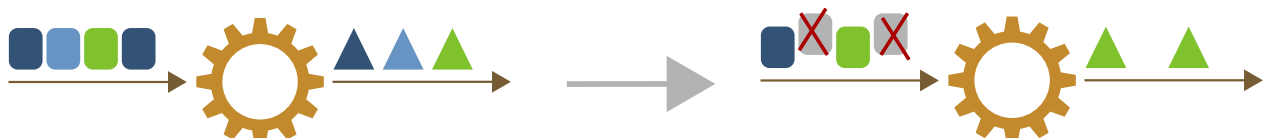


Valeria Cardellini - SABD 2023/24

8

Challenge 3: Manage load variations

- Some solutions:
 - Admission control
 - Static reservation
 - Reserve specific resources in advance
 - *Cons*: over-provisioning and cost increase
 - Apply dynamic techniques such as **load shedding**
 - Selectively drop tuples at strategic points (e.g., when CPU usage exceeds a specific limit)
 - *Cons*: sacrifice accuracy and completeness

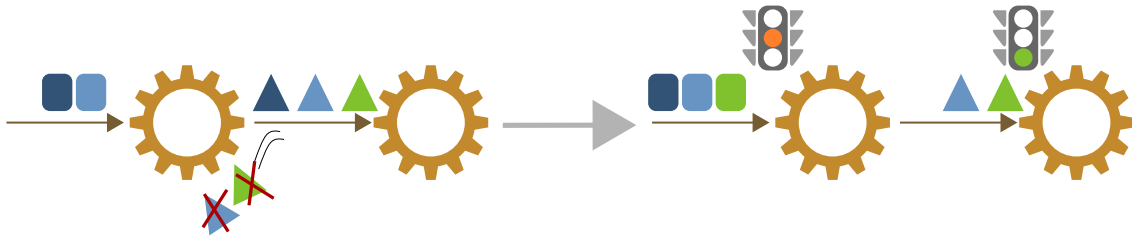


Valeria Cardellini - SABD 2023/24

9

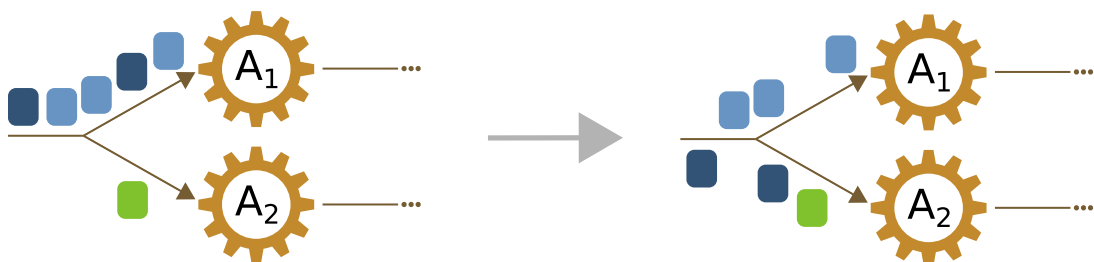
Challenge 3: Manage load variations

- Some solutions (*continued*):
 - Use **adaptive rate allocation: backpressure**
 - The upstream operator that precedes the bottleneck operator stores data in an internal buffer to reduce the pressure
 - Backpressure can recursively propagate up to the source operators



Challenge 3: Manage load variations

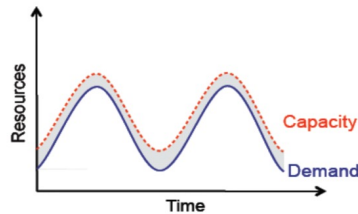
- Some solutions (*continued*):
 - Redistribute load, e.g., by changing stream partitioning or determining a new operator placement and relocating operators on computing nodes
 - *Cons*: available resources could be insufficient



Exploit elasticity



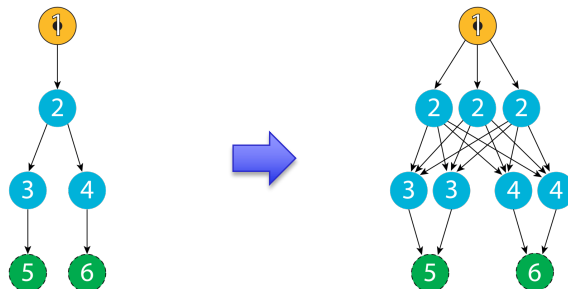
- Another solution:
 - Detect bottleneck and solve it by exploiting **elasticity**: acquire and release resources when needed



- *How?*
 - By hand: possible, but cumbersome
 - So what? Self-adapt and organize the architecture using MAPE!

Elastic data stream processing

- *Where?*
 - At **application layer** (i.e., operator scaling)
 - i.e., apply **SPMD** paradigm: concurrent execution of multiple replicas of the same operator on different data portions
 - Scale-out (in) operators by adding (removing) operator replicas



Elastic data stream processing

- *Where?*
 - At **infrastructure layer**
 - Scale horizontally computing resources (containers, virtual machines, physical machines)
 - Also scale vertically computing resources (containers, virtual machines)

Elastic stream processing

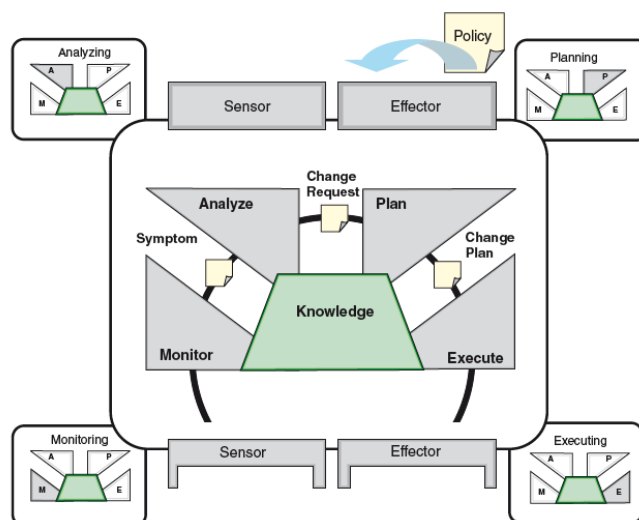
- *When* and *how* to scale?
 - Open issues
 - Some simple example:
 - When: threshold-based (like AWS Auto Scaling)
 - How: add/remove one operator replica at time
 - Where: determine randomly (or in a round-robin fashion) location of new replica
- Be careful: elasticity overhead is not zero!
 - In most streaming systems: required to run new placement decision to take new replicas into account
 - Dynamic scaling impacts stateful operators

Challenge 4: Self-adapt at run-time

- Many factors may change at runtime, e.g.,
 - Load variations, QoS of computing resources, cost of computing resources (e.g., due to dynamic pricing schemes), network characteristics, node mobility, ...
- How to adapt the DSP application when changes occur?
 - Enrich DSP systems with run-time adaptation capabilities
- Which **adaptation actions**?
 - Scale-out/in number of operator replicas
 - Migrate operators on different computing nodes
 - ...

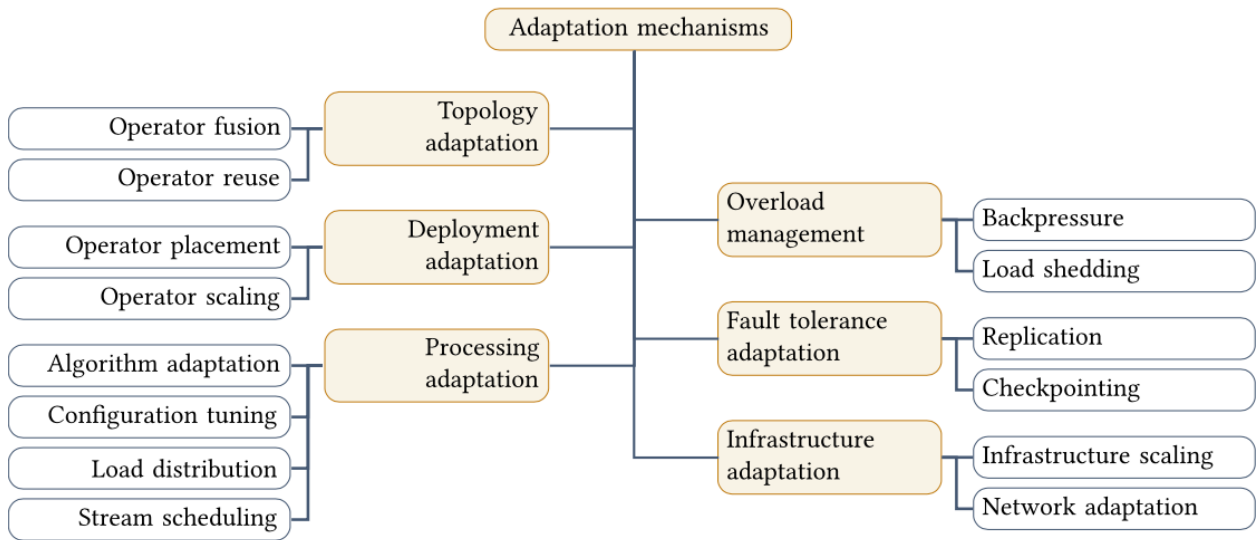
Self-adaptive deployment

- **MAPE** (**M**onitor, **A**nalyze, **P**lan and **E**xecute)

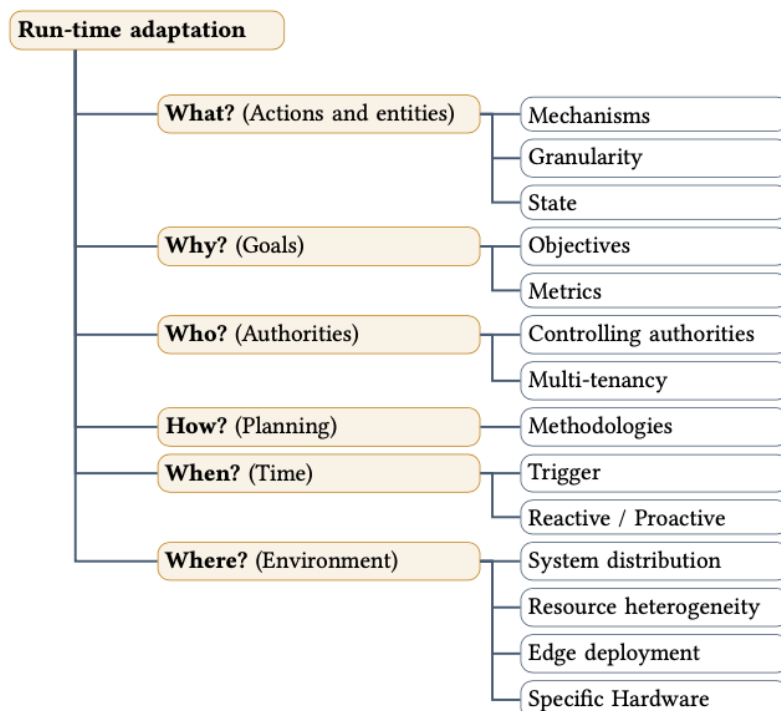


- **Plan** phase: how to **adapt** DSP application deployment

Main adaptation mechanisms



Dimensions used to classify adaptation solutions



Reconfiguration challenges

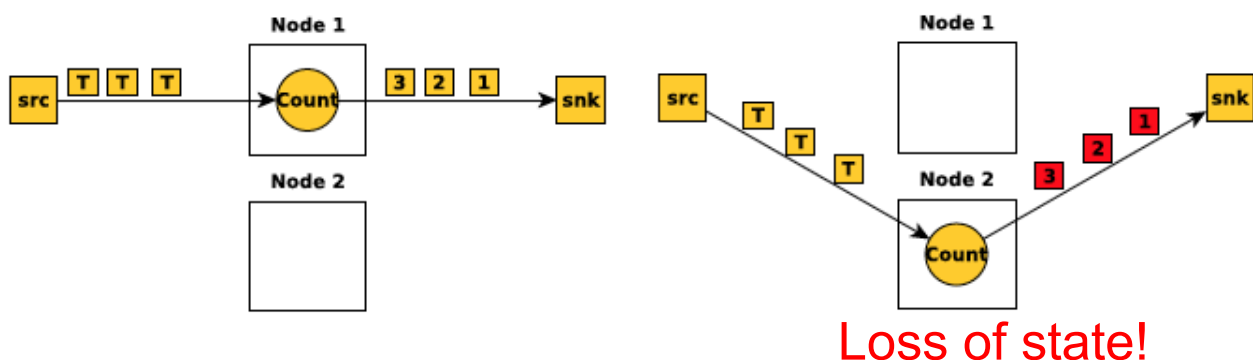
- Deployment reconfiguration has a non-negligible cost
- Can affect negatively application performance in the short term
 - Application freezing times caused by operator migration and scaling, especially for **stateful** operators
- Solution:
 - Perform reconfiguration only when needed
 - Take into account the overhead for migrating and scaling the operators

Challenge 5: Stateful operators

- State complicates things...

1. Dynamic scaling
2. Operator re-placement
3. Recovery from failure

impact state



Approaches for stateful migration

- Most streaming systems do not support migration of stateful operators
- Some research prototypes and production-ready streaming systems do
 - E.g., Spark Streaming
- Requirements for stateful operator migration
 - Safety (i.e., to preserve operation consistency)
 - Application transparency
 - Minimal footprint

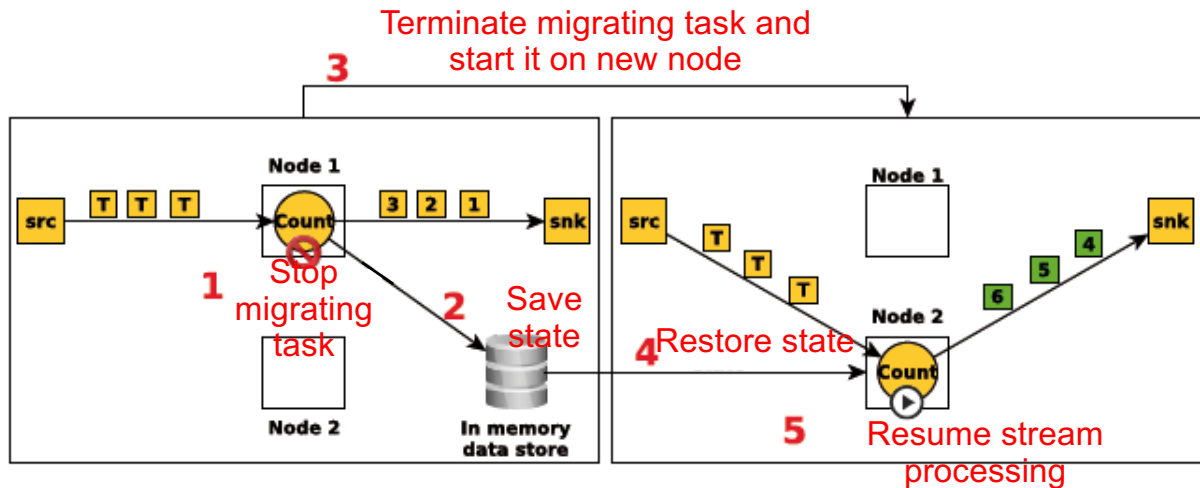
Issues with stateful operators

- Require mechanisms to:
 - Migrate stateful operators
 - Pause-and-resume approach
 - Parallel track approach
 - Partition streams and load balance among operator replicas

Stateful operator migration

- Pause-and-resume approach

👎 Application latency peak during migration



Stateful operator migration

- Parallel track approach

- Old and new operator instances run concurrently until their state is synchronized

👍 No latency peak

👎 Enhanced mechanisms for synchronization

Issues for stateful migration: stream partitioning

- How to identify which portion of state to migrate? Possible approaches:
 - Expose an API to let the user manually manage the state
 - Support only **partitioned stateful operators**
 - Partitioned stateful operators store independent state for each sub-stream identified by a partitioning key
 - Automatically determine, on the basis of a partitioning key, the optimal number of state partitions to be used and migrate

Issues for stateful migration: load balancing

- How to balance the load among multiple stateful replicas?
- Can use consistent hashing
- Can use partial key grouping
 - Uses two hash functions where a key can be sent to two different replicas instead of one
- Only available in research prototypes

Challenge 6: Guarantee fault tolerance

- DSP applications run for long time
 - ➡ failures are unavoidable
- Possible solutions:
 - Active replication
 - Check-pointing
 - Replay logs
- Having different trade-offs between runtime cost in absence of failures and recovery cost
- Large-scale complicates things...
 - Network partitions and CAP theorem

References

- M. Hirzel, R. Soulé, S. Schneider, B. Gedik, R. Grimm, [A catalog of stream processing optimizations](#), *ACM Comput. Surv.*, 2014
- V. Cardellini, F. Lo Presti, M. Nardelli, G. Russo Russo, [Run-time adaptation of data stream processing systems: The state of the art](#), *ACM Comput. Surv.*, 2022