Challenges in Data Stream Processing

Corso di Sistemi e Architetture per Big Data
A.A. 2016/17

Valeria Cardellini
Challenge 1: Optimize the DSP application

• Apply some transformation to streaming graph
  – At design time or run-time

• Operator reordering
  – To avoid unnecessary data transfers

• Redundancy elimination
Challenge 1: Optimize the DSP application

- Operator separation

- Fusion
Challenge 2: Place the operators

- Operator placement decision: a 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
    - Move only relocatable operators
    - Require operator migration
- We will focus on this issue in a next lesson
Challenge 3: Manage load variations

• Typical stream processing workloads are:
  – with high volume and high rates
  – bursty and with workload spikes not known in advance
    • Twitter in 2013: rate of tweets per second = 5700
    • … but significant peak of 144,000 tweets per second
Challenge 3: Manage load variations

• Possible approaches:
  – 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
Challenge 3: Manage load variations

• Possible approaches (continued):
  – Use adaptive rate allocation
    • E.g., backpressure: the upstream operator that precedes the bottleneck stores data in an internal buffer to reduce the pressure; backpressure recursively propagates up to the source operators
  – Redistribute load, e.g., determine new operator placement and relocate operators on computing nodes
    • Cons: available resources could be insufficient

• What else?
Exploit data parallelism

• Alternative solution:
  – Detect bottleneck
  – Use data-parallelism (aka operator fission)
    • Apply SIMD paradigm: concurrent execution of multiple replicas of the same operator on different data portions
    • By hand: possible, but cumbersome
Elastic stream processing

• Exploit **elasticity**: acquire and release resources when needed

• **Where?**
  – At application layer (i.e., data parallelism)
    • Scale out (or scale in) operators by adding (removing) operator replicas
    • Activate (or deactivate) already replicated operators
  – At infrastructure layer
    • Scale out (or scale in) computing nodes
Elastic stream processing

• *When* and *how* to scale?
  – Open issues that deserve investigation
  – Some simple example:
    • *When*: threshold-based
    • *How*: add/remove one replica at time, but where to place it?

• Be careful: elasticity overhead is not zero!
  – In most streaming systems: run a new placement decision to take the new replicas into account
  – Dynamic scaling impacts stateful operators
Challenge 4: Self-adapt at run-time

• To cope with highly dynamic operative environment
  – Unpredictable workload
  – Computational characteristics of operators not known a-priori
  – Need to sustained load for long provisioning times
  – Node availability, network congestion, …

• Exploit run-time adaptation capabilities of DSP systems

• What adaption actions?
  – Scale the number of operator instances, relocate the operators, …
Self-adaptive deployment

- **MAPE** (Monitor, Analyze, Plan and Execute)

- **Plan** phase: how to **reconfigure** the application deployment
Distributed Storm

- We developed an extension of Storm
- Goals: to provide
  - distributed monitoring
  - distributed placement
  - and adaptation capabilities
- Where: large-scale environment
- Code available on GitHub
  matnar.github.io/uniroma2-storm/

Distributed Storm architecture
Distributed Storm: monitoring

- **QoSMonitor** (for each worker node)
  - Estimate network latencies
    - Use a network coordinate system
    - Vivaldi’s algorithm: decentralized and gossip-based
  - Monitor QoS attributes
    - Node utilization and availability

- **Worker Monitor** (for each worker process)
  - Monitor exchanged data rate among the operators
Distributed Storm: performance

Load spike on a subset of nodes

~50%
Reconfiguration challenges

• Reconfiguring the deployment 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

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

Loss of state!
Approaches for stateful migration

• Most of streaming systems do not support stateful processing and migration (e.g., Storm)
  – Developers manage state
  – Typically combine with external system to store state
  – Design complexity

• Requirements for stateful operation migration
  – Safety (i.e., to preserve the consistency of the operations)
  – Application transparency
  – Minimal footprint
Stateful operator migration

• Two approaches:
  – Pause-and-resume
  – Parallel-track

• Pause-and-resume approach

1. Stop migrating task
2. Save state
3. Terminate migrating task and start it on new node
4. Restore state
5. Resume stream processing
6. In memory data store

Valeria Cardellini - SABD 2016/17
Stateful operator migration

• Pause-and-resume drawback
  – Peak in the application latency during the migration

• Parallel-track approach
  – Old and new operator instances run concurrently until
    the state of both is synchronized and the new
    instance can safely take over
  – Drawback: requires enhanced mechanisms

• No clear winner
Issues for stateful migration

• How to identify the portion of state to migrate?
  – 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

• 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
Elastic stateful migration in Storm

- We developed mechanisms for elastic stateful migration in Storm


Valeria Cardellini - SABD 2016/17
Elastic stateful migration in Storm

• Scaling decisions at the framework level
  – Adapt the number of parallel instances for each application operator
  – Simple threshold-based scaling policy

• Relocate the operator internal state on a different node and enable Storm to change the application deployment at run-time
Performance results

- DSP app: frequent pattern detection

- Elastic scaling and stateful migration improves the application latency
Challenge 6: guarantee fault tolerance

• DSP applications run for long time intervals failures are unavoidable

• Possible solutions:
  – Active replication
  – Check-pointing
  – Replay logs
  – Hybrid solutions

• Having different trade-offs between runtime cost in absence of failures and recovery cost

• Large-scale complicates things…
  – Network partitions and CAP theorem
References


• V. Cardellini, M. Nardelli, D. Luzi, “Elastic stateful stream processing in Storm”, *Proc. HPCS 2016*.