Imperial College London

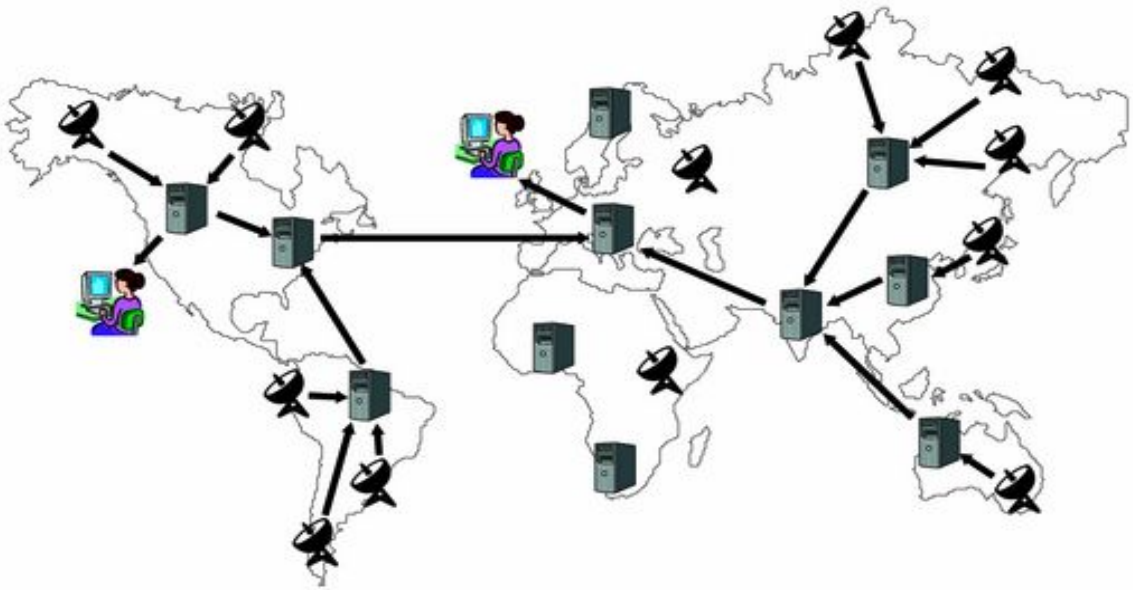
A Quality-Centric Data Model for Distributed Stream Management Systems

Marco Fiscato mfiscato@doc.ic.ac.uk
Quang Hu Vieu qhvu@doc.ic.ac.uk
Peter Pietzuch prp@doc.ic.ac.uk

Distributed Software Engineering Section
Department of Computing

Distributed Stream Management Systems (DSMS)

- Allow the processing of queries over data streams in *real-time*.
- *Sources* are geographically *distributed* over the world *(i.e. sensors)*.
- More scalability by *distributing computation* over several data centers.



Applications

- **DSMS** can be used in many application domains:
  - Environmental monitoring
  - RFID tag tracking
  - RSS feeds aggregation
  - Internet monitoring

- Example: **Meso-scale weather monitoring**

  (*Severe storms / Hurricanes / Tornadoes)*

  Early Forecast → Impact Mitigation

Stream processing can be used for accurate real-time weather prediction

Dependability

- We consider a large DSMS deployed at **Internet-scale**.
- A shared infrastructure with *multiple input providers* (i.e. sensor networks).
- **Resource constraint environment** (over provisioning is infeasible).
- In general **failure** is going to be the *common case:*
  - *Host failure*
  - *Network partitions*
  - *CPU overload*
- **Impossible** to achieve **perfect processing**.

In many cases an imperfect result
is better than no result at all.

A Quality-Centric Data Model

- The system needs to operate under **constant partial failure.**
- Users need to accept *incomplete* or *inaccurate* results.

➔ **We propose a <u>Quality-Centric Data Model:</u>**

  - ➔ to help the *query optimisation* and *fault recovery.*
  - ➔ to report the achieved *quality of service.*

- Enhance tuples with three new **quality metrics**:

  - ➔ ***Weight:*** measures the *"importance"* of data
  - ➔ ***Recall:*** measures the *"quality"* of the data
  - ➔ ***Utility:*** a *combination* of the weight and recall

Talk Outline

- Overview
- Dependability issues
- Quality-centric data model
- Improving fault-tolerance
- Evaluation
- Future work
- Conclusion

Design considerations

- Focus on **quality of the processing**, not quality of the data.
  - *Dependability*

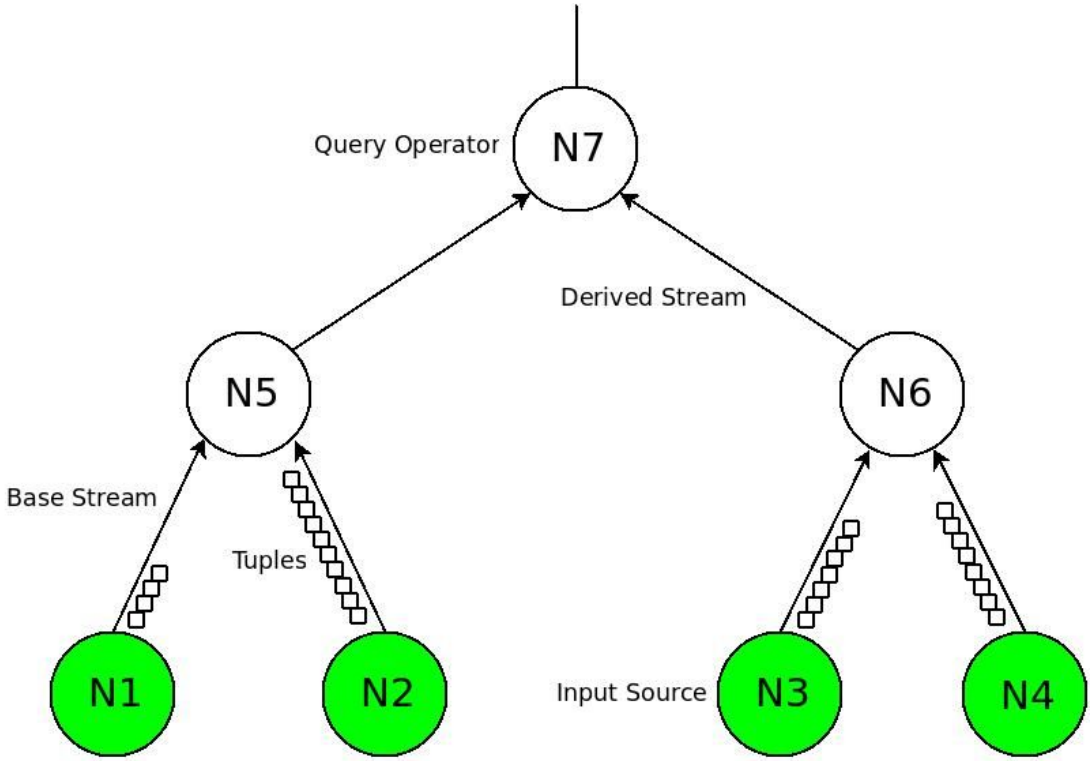
- Should support the *broadest set of operators*:
  - **Operator agnostic** approach *(ignore semantics)*

- Metrics should help the system in:
  - Fault recovery / Query optimization
  - Also measuring the achieved Quality-of-Service.

- Possible metrics:
  - *Confidence* → need to understand operator semantics (i.e. 20mph 90% prob.)
  - *Error Range* → difficult to establish (20mph ± 10%)

Need for more generic quality metrics

Stream Processing Model



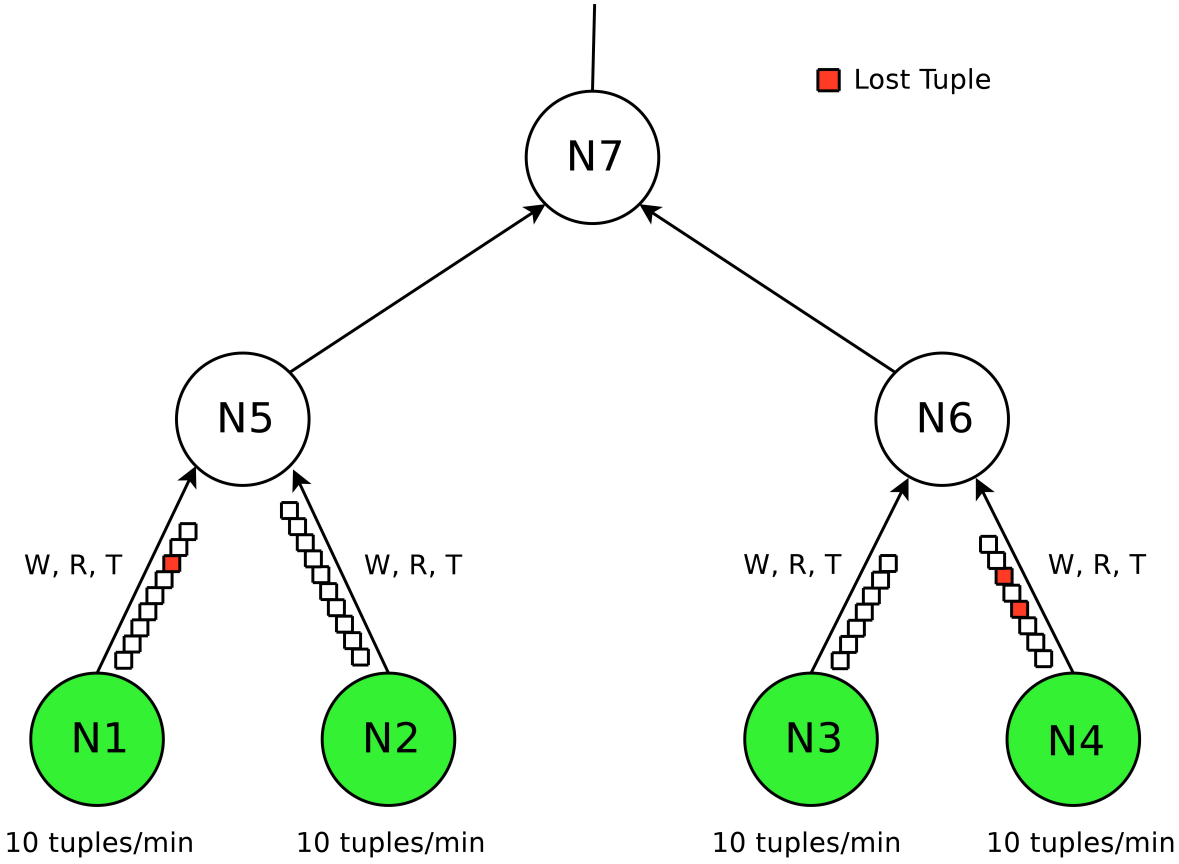
A simple query plan.

The Quality Metrics

Three **quality metrics** to establish the *quality of the processing*:

- **Weight**: *how many sources (units)?*
  - Measures the "importance" of data
  - Grows as streams are combined
- **Recall**: *how much information is captured by the result?*
  - Measures the "quality" of the data
  - If lower than 1 signal data loss
- **Utility**: *how important is it overall?*
  - Generated from recall and weight
  - Most important metric of all

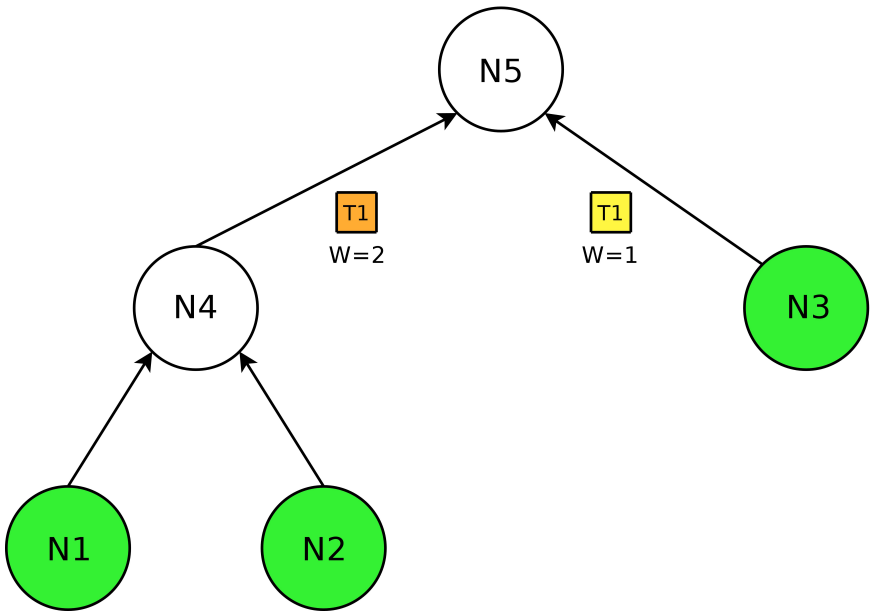
A Simple Query



A simple query plan **with quality metrics**

Quality Metrics: Weight

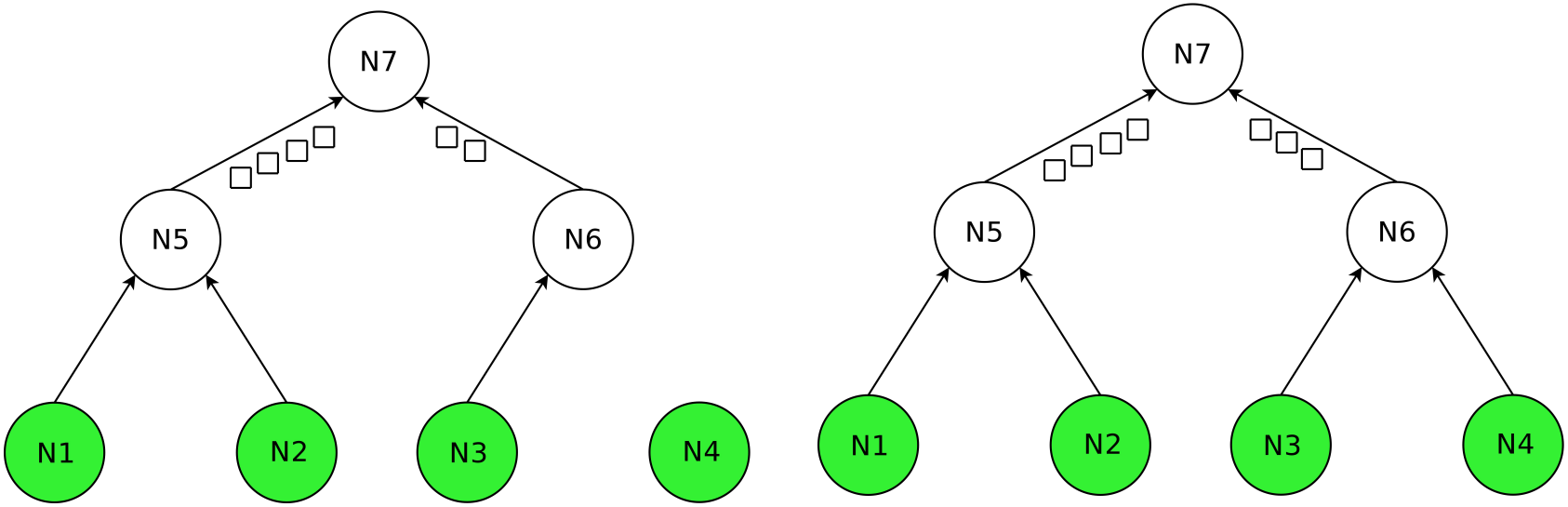
Weight helps dealing with *unbalanced* query plans.



- Tuples in *base streams* have all **weight** *equal to 1*.
- Can be *increased if tuple* is considered *more important*.
- Tuples become *heavier as they climb up* the query plan.

Quality Metrics: Recall

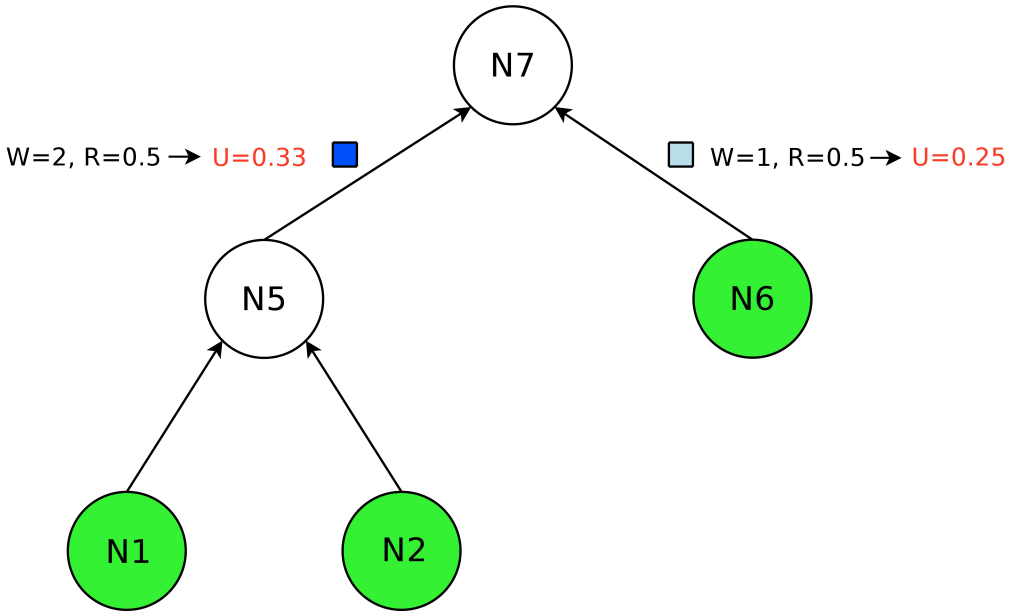
Recall helps *quantifying losses.*



- Amount of information captured by the tuple / total possible with no failure.
- Can be *lowered by failure* (crash, load-shedding, …).
- Can be used to *estimate the impact of failure.*

Quality Metrics: Utility

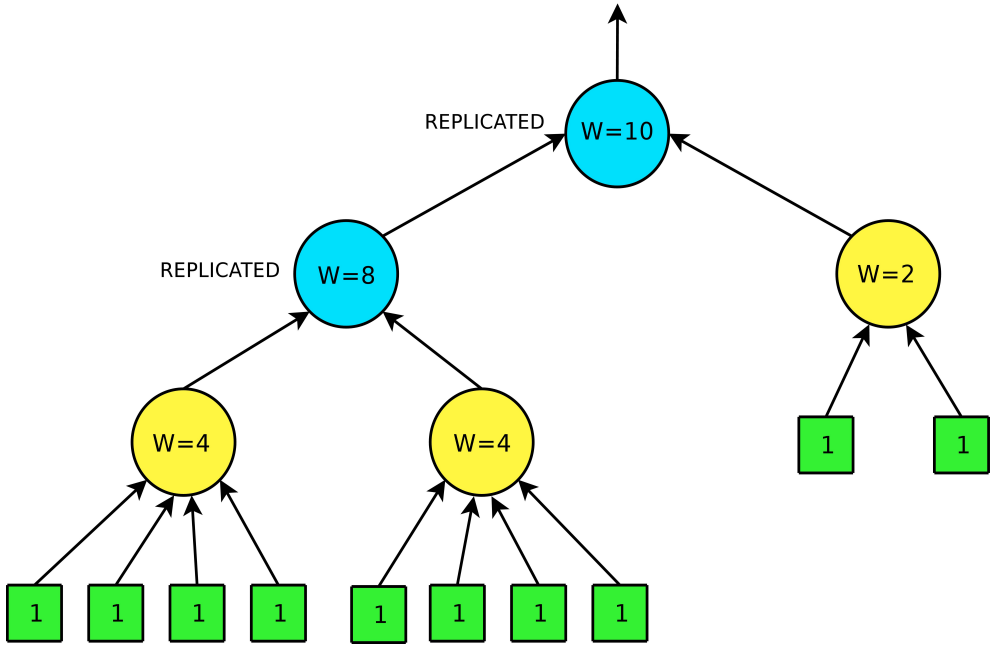
Utility *puts importance into perspective*



- *Normalize* recall by weight.
- In absence of failure, *always increases.*
- Useful for *replication* and *load-shedding.*

Proactive Replication

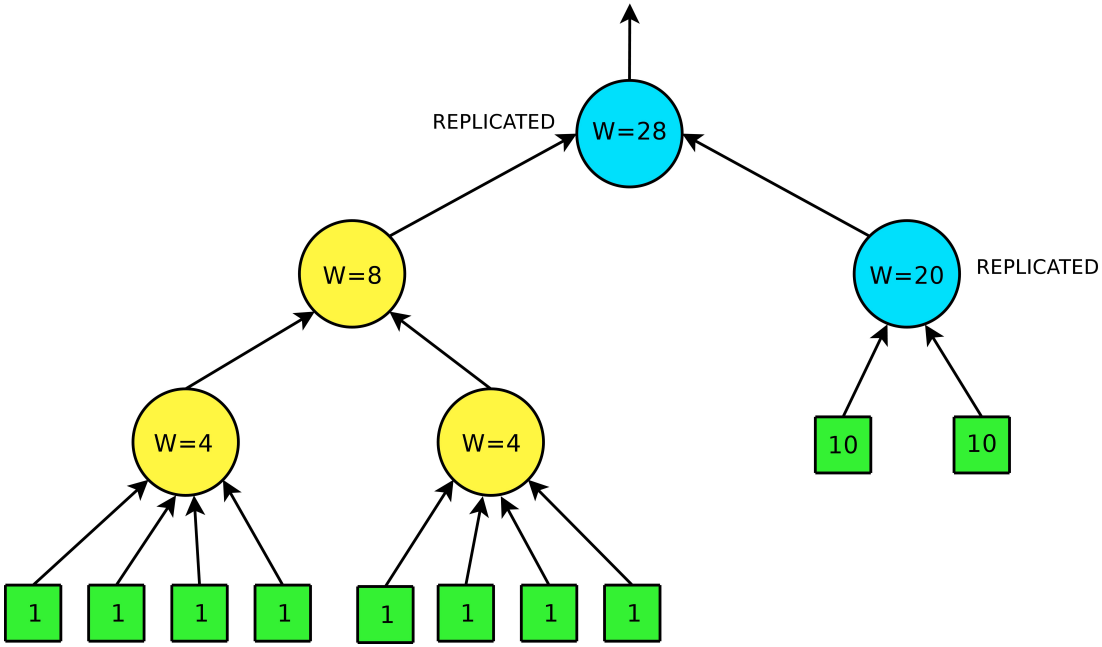
Weight can be used for **proactive replication** of operators



Always replicate operators processing "heavy weight" streams first.

Proactive Replication

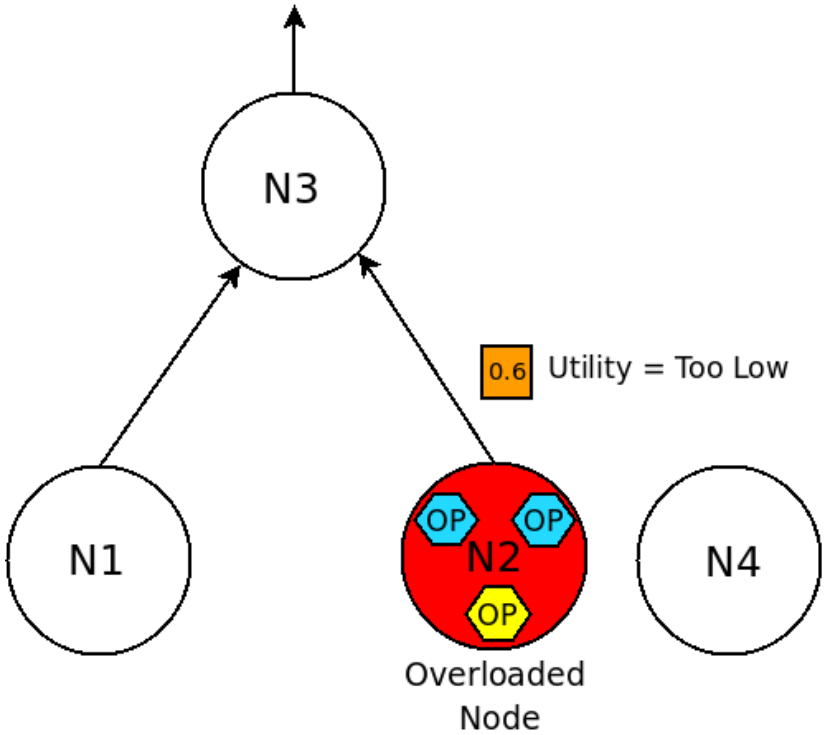
Weight can be used for **proactive replication** of operators



Always replicate operators processing "heavy weight" streams first.

Operator Migration

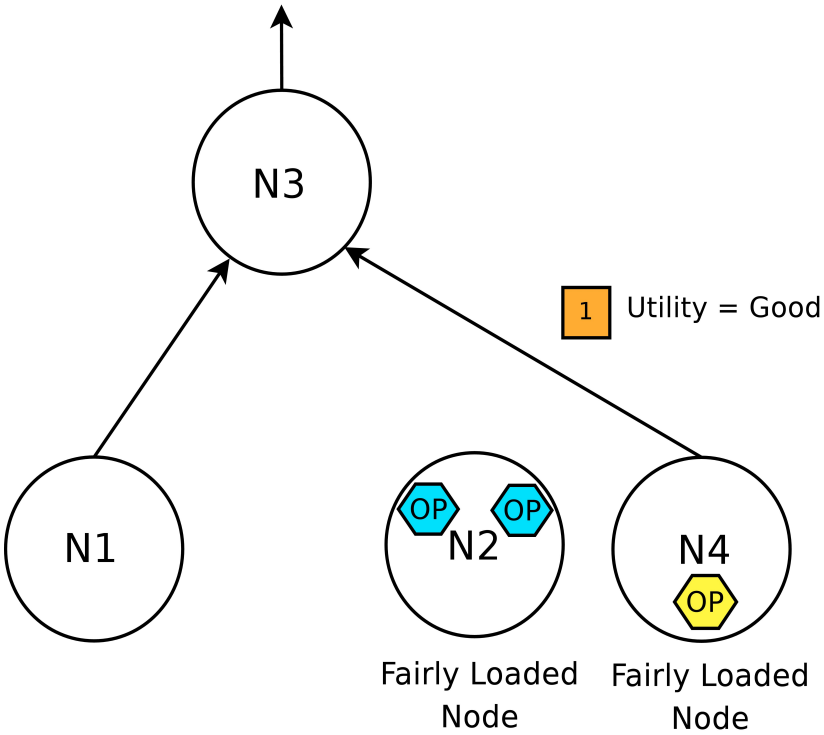
Utility can be used to detect *overload conditions*



N2 becomes overloaded → low utility of output

Operator Migration

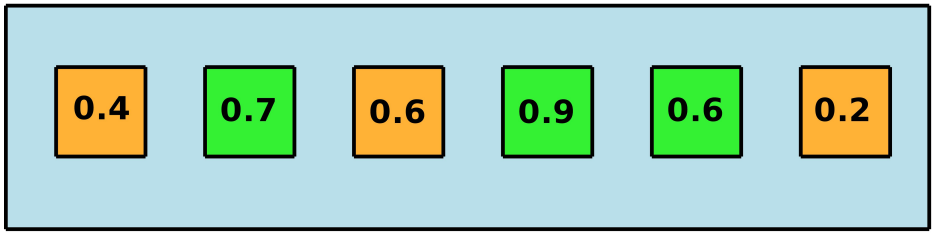
Utility can be used to detect *overload conditions*



Moving the heavy operator to N4 increases the utility of output.

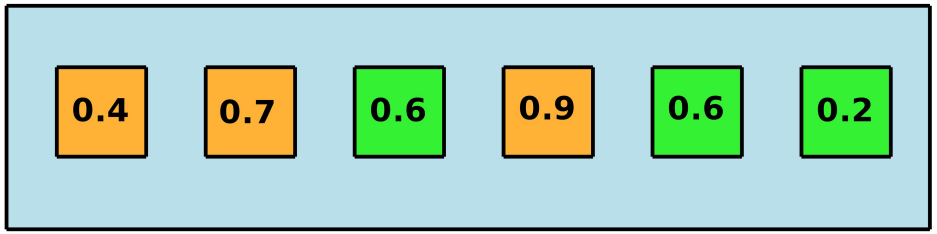
Quality-aware Load-shedding

Quality-aware load-shedding



Output utility: 0.55

Random load-shedding



Output utility: 0.425

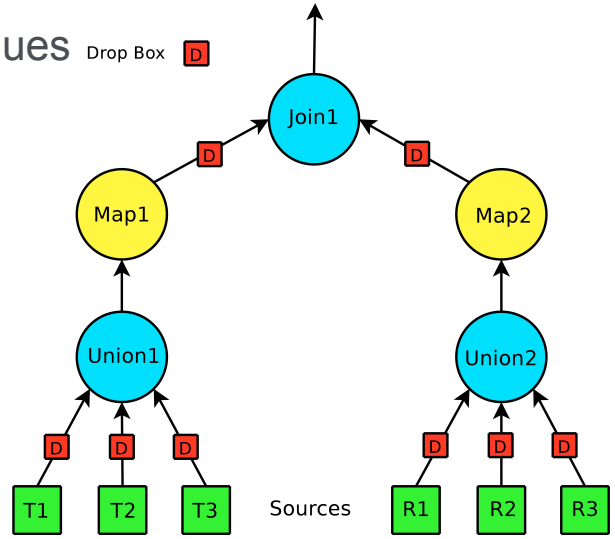
(Weight =1, Tot. Sources = 4)

Intelligent load-shedding helps improving the *utility* of tuples.

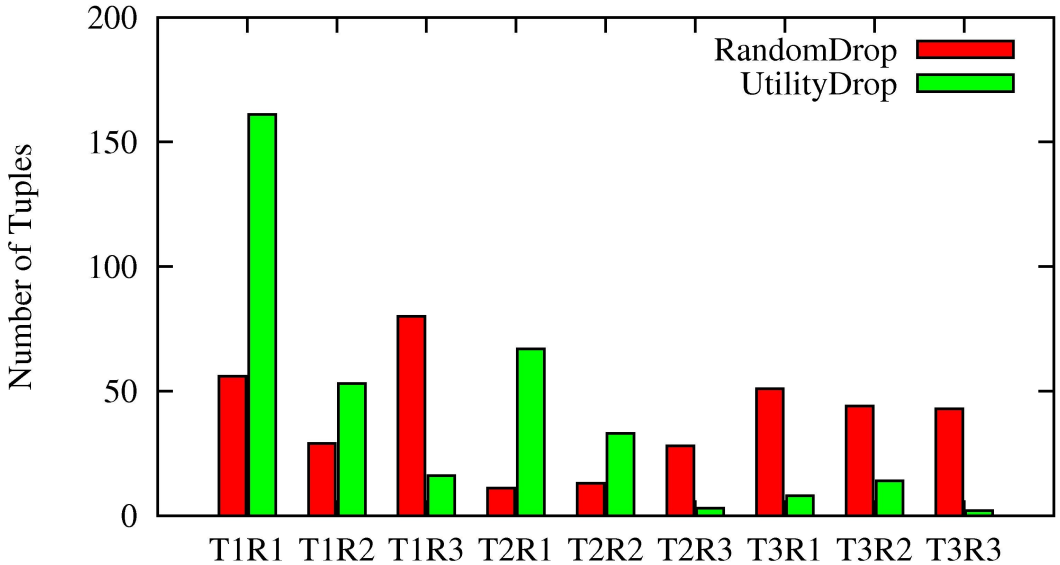
Borealis Extension

- For evaluation, implemented in Borealis
- Modified XML file describing query:
  - add recall and weight to the tuples
  - add operators when necessary
- We can distinguish 3 classes of operators:
  - **Map/Filter** → only add extra fields
  - **Join** → add extra fields and calculation of new values
  - **Aggregate** → add extra fields and a map box
- Latency overhead:

Map	1%
Join	1%
Aggregate	18%



Evaluation



Distribution of tuple pairs

Quality-aware load shedding:
can improve data quality in results for certain types of queries.

Example:
Redundant sensors for reliability. (temperature and rain)

Only consider sensor couple with the largest number of readings.

Quality-aware load-shedding helps in skewing the distribution of results.

Higher number of tuples is better.

Future Work

- Evaluate the model in real world deployments.
- Investigate replication policy based on the quality model.
- Test how results are affected in different types of queries.
- Experiment with other quality metrics.

Conclusion

- Stream processing can be used for a variety of *monitoring applications.*
- When deployed at Internet-scale *failure is not avoidable*.
- **Quality-centric data model** to estimate the quality of processing.
- Metrics are *generic* and *independent of operator semantic.*
- Allow for *intelligent load-shedding.*
- *Useful for improved fault-tolerance* → ***dependability***.