Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik
Computer Science

Staying FIT with Aurora/Borealis

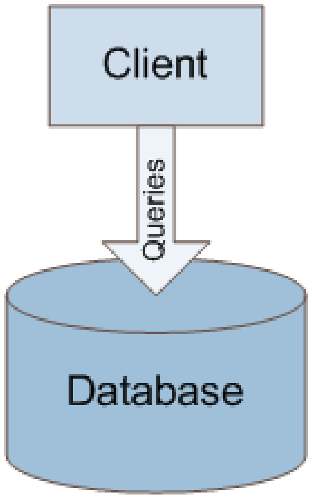
Wednesday, 01 October 2008

Overview

- Introduction to Stream Processing
- Aurora
- Borealis
- FIT
- Summary and Trends

INTRODUCTION

Classic Database

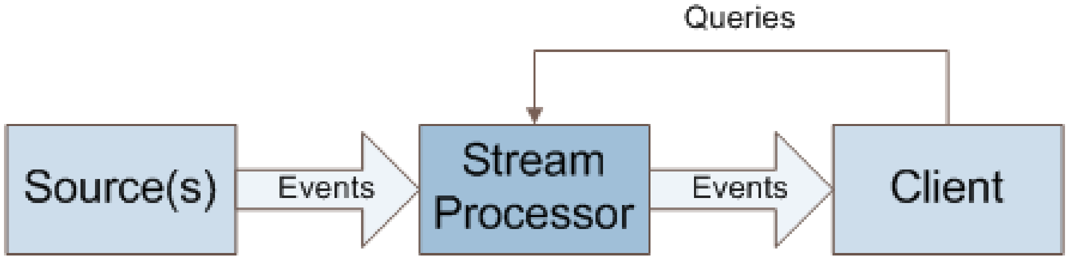


- Database
  - A large, mainly static collection of data
  - Contains the last, *current* state of data
    - Notion of time and history difficult to encode
- Human-Active, DBMS-Passive (HADP)
  - Database sits and waits for queries
  - Queries actively *pull* out data
  - Precise answers, no notion of real-time

Problems?

- Sensor monitoring, financial analysis, …
  - Continuous *streams* of data
    - Stock quotes, RFID tags, business transactions
  - Long running, *continuous* queries
    - "Alert me when share price falls below $1…"
  - Queries over history or time windows
    - "… and does not recover within 10 minutes."
- Classic DBMS inadequate
  - Triggers not suitable for high update rates and history
  - Cf.: Stonebraker's "One Size Fits All…" papers

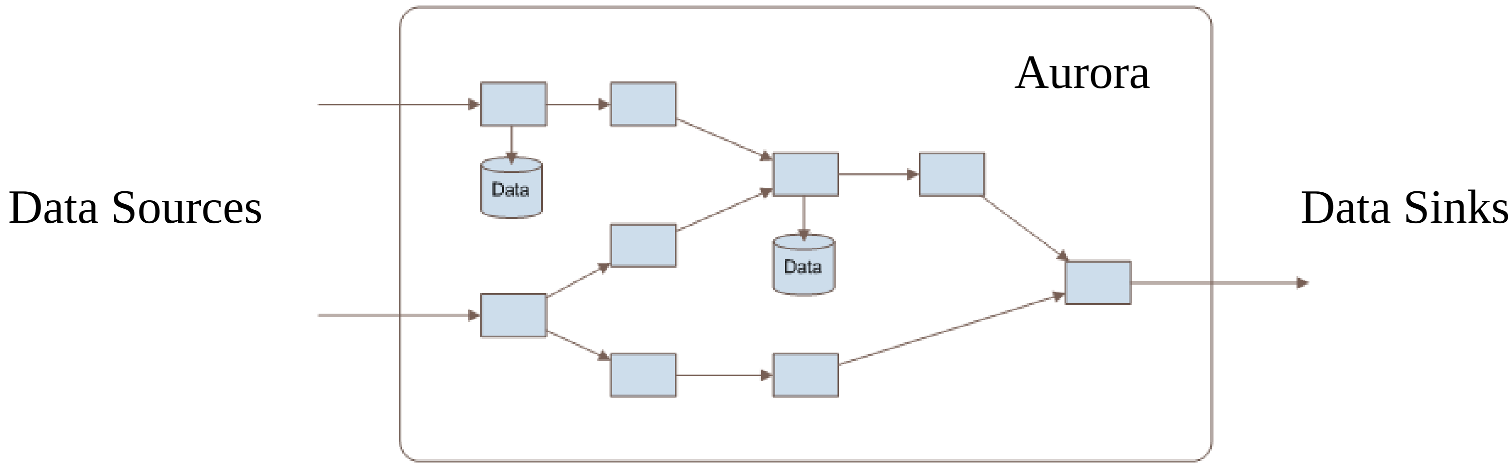
Stream Management System



- DBMS-Active, Human-Passive
  - Analogous to publish-subscribe systems
- Designed for monitoring applications
  - Complex queries over high-volume streams
  - Real-time response favored over answer precision
  - Time and sequence integral to data model

AURORA

System Model



- Centralized data-flow system
  - “Boxes and arrows” paradigm
  - Data sources *push* tuples through an operator network
  - Supports multiple input and output streams

Query Model

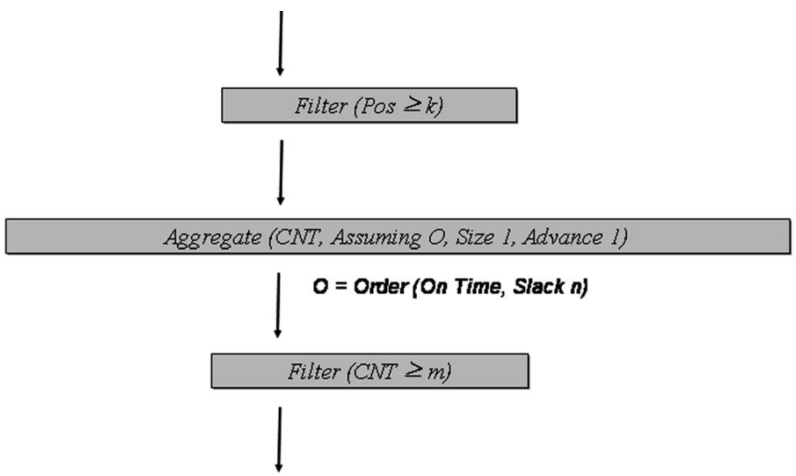
- Supports continuous and ad-hoc queries
  - Specified as operator "box" networks by the admin
  - "Arrows" are implemented as disk-resident queues
  - Output arrows have QoS-specifications
    - Basis for scheduling and load-shedding decisions
- Connection points
  - Located on selected arrows
  - Allow extension of network and persistent storage
    - Static data sources and history buffering

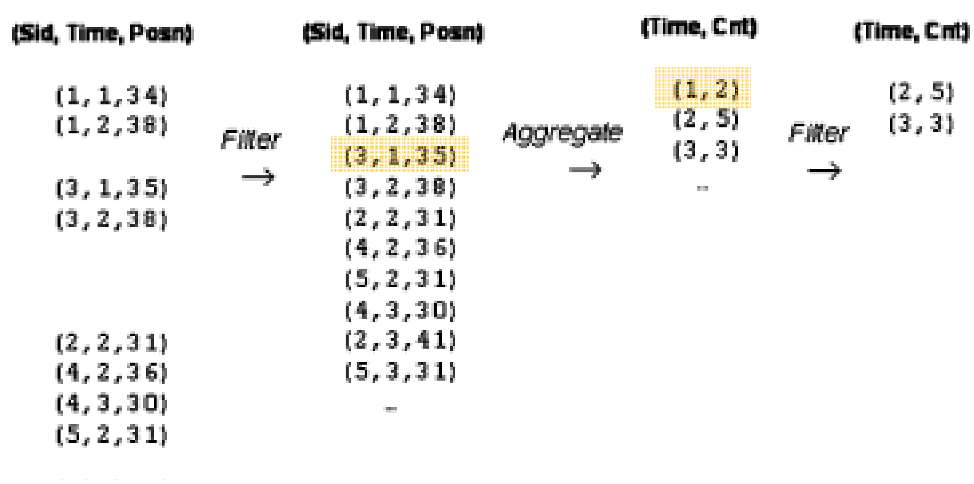
Operators

- Order-agnostic operators
  - Filter, Map, Union
  - Operate tuple-wise on infinite streams
- Order-sensitive operators
  - BSort, Aggregate, Join
  - Operate on sliding, (semi-)ordered *windows*
    - Finite sequences of consecutive tuple arrivals
    - Specified as length of sequence and/or physical time-span

Query Example

- Stream schema: Soldier(Sid, Time, Posn)
- "Produce an output whenever m soldiers are across some border *k* at the same time, where "across" is defined as *Posn* ≥ *k*"

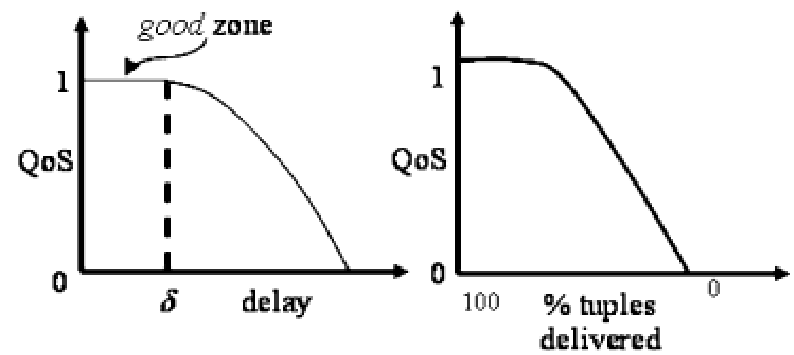




let m = 5, k = 30, n = 1

Load Shedding

- Static analysis
  - Test feasibility based on expected arrival rates, tuple processing cost, and operator selectivities
- Dynamic load monitoring
  - Monitor QoS at outputs
    - QoS requirements specified as monotonic utility functions



  - If not: use gradient walk to find most tolerant output
    - Then go "upstream" and insert drop operators as early as possible

BOREALIS

Feature Overview

- Successor to Aurora
  - Messages may be inserts, updates, or deletes
    - Aurora supported only inserts (“append-only” solution)
    - Allows *data revision* after the fact
  - Dynamic query modification
    - Users may specify conditional plans and operator attributes
  - Distributed system
    - Aimed at “sensor-heavy, server-heavy” use cases
    - Higher scalability and fault-tolerance

Revision Messages

- Allow recovering from mistakes
  - E.g. “Sorry I gave you the wrong stock quote earlier, here is the real one”
  - Problem: Revision messages are expensive!
    - Implemented by *replaying* the history and propagating the delta
    - Requires storing the history of every operator
    - Particularly expensive for stateful operators (e.g. aggregate)
- Used to implement *time travel*
- Used for Borealis’ replication scheme

Optimization

- Load shedding and operator placement
- Local
  - Similar to Aurora but with different QoS model
- Distributed
  - Global (centralized), and neighborhood (peer-to-peer)
    - Move operators between nodes
  - Unclear relationship to fault-tolerance
    - What if the global optimizer fails?
    - Consensus between replicas on operator placement?

Fault-Tolerance

- Replication
  - Idea: *SUnion* operator *deterministically* serializes input from multiple upstream replicas
  - Output is multi-casted to any downstream replicas
  - Eventual consistency
    - Finite logs, messages may get lost
    - Revision messages for reconciliation
    - Good enough since clients do not expect precise answers anyways
- Loose ends
  - Permanent node failure not handled
  - Single points of failure (global optimizer and global catalog)
  - What about neighborhood optimization?

Scalability

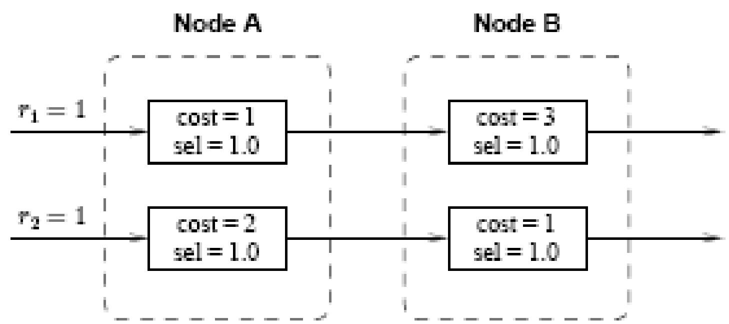
- Vision of massive, hierarchical federations
  - *Regions* of nodes treat each other as virtual nodes
  - Hierarchical optimization based on SLAs
- Ideas seem a bit over-ambitious at this point
  - No mechanism for adding/removing nodes at runtime
    - (Generalization of the permanent node failure problem)
  - A *lot* of critical system state to replicate
    - Global catalog, optimization decisions
    - Especially if nodes can come and go…

FIT

Overview

- Off-line, distributed load shedding algorithm
  - Plans for different load scenarios created up front
  - Considers only CPU cost and a single utility metric
- Plugin for Borealis
- FIT = “Feasible Input Table”
  - Name of the main data structure in algorithm
- Designed for internet-scale sensor networks (?)

Problem Description

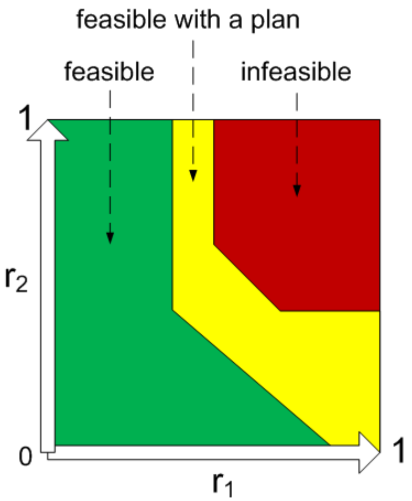


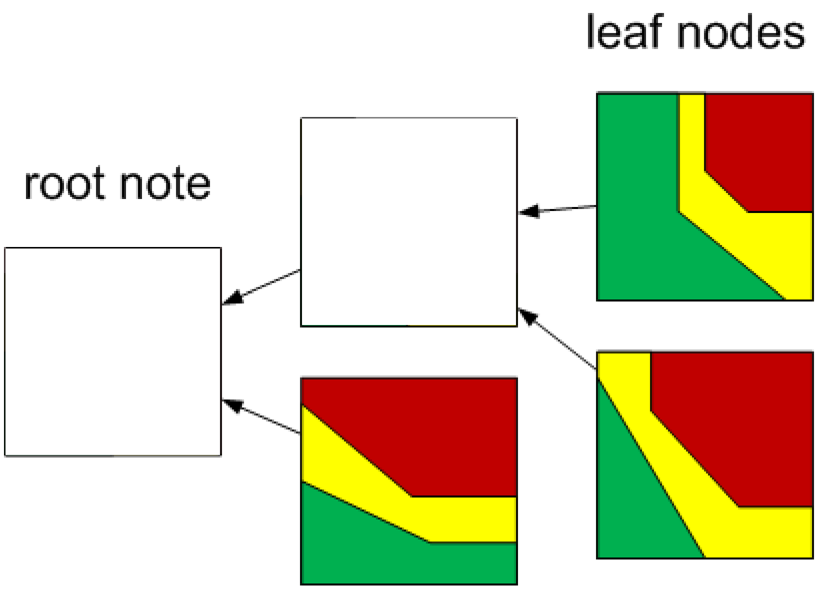
Plan	Reduced rates at A	A.load	A.throughput	B.load	B.throughput	Result
0	1, 1	3	1/3, 1/3	4/3	1/4, 1/4	originally, both nodes are overloaded
1	1/3, 1/3	1	1/3, 1/3	4/3	1/4, 1/4	B is still overloaded
2	1, 0	1	1, 0	3	1/3, 0	optimal plan for A, but increases B.load
3	0, 1/2	1	0, 1/2	1/2	0, 1/2	both nodes ok, but not optimal
4	1/5, 2/5	1	1/5, 2/5	1	1/5, 2/5	optimal

- Optimization problem
  - Maximize the weighted score of outputs under linear load constraints
  - Can be solved exactly by *linear programming*
    - Baseline for performance comparison by the paper

The FIT Approach

- Meta-data aggregation and propagation from leaf nodes to the root node
  - Meta-data = Feasible Input Table (FIT)
  - A set of feasible input rate combinations





Results

- Paper describes efficient heuristics to compute and merge FITs
  - 3 orders of magnitude faster than linear programming
- What is *efficient*?
  - Runtime and size of FIT is *exponential* in the number of inputs
  - Impractical for more than a few loosely linked nodes and inputs (≤ 5)

Limitations

- Limited to one resource (CPU)
  - Model assumes that twice the input equals twice the work
  - But: per-tuple cost is non-linear as shown by Aurora
- Considers append (insert) events only
  - What happened to Borealis' revision messages?
- Nodes form an immutable tree topology
- Operator network may not change
  - Otherwise re-plan up the stream starting from point of change
  - Neighborhood optimization?
- Does not scale beyond a few nodes and inputs

SUMMARY AND TRENDS

Summary

- Aurora
  - A centralized stream management system with QoS-based scheduling and load shedding
- Borealis
  - A distributed stream management system based on Aurora
  - Adds revision events and fault-tolerance
- FIT
  - An off-line, distributed load shedding algorithm
  - Too limited and impractical (in current form)

Critique and Trends

- Borealis research increasingly esoteric
  - Lack of use cases for “internet-scale” networks
  - Lack of use cases for sophisticated load shedding
  - But: Multi-core trend creates potential for similar approaches at a local level

Critique and Trends (2)

- Real money lies in *integrating* stream processing with large data stores
  - Business Process Monitoring
  - Database integration in Borealis is insufficient
    - True for any existing streaming system
  - SAP and Oracle are spending *billions* on it
  - ADMS group at ETH now focuses on this topic

References

- *Aurora: a new model and architecture for data stream management*, Abadi et. al, VLDB Journal 12(2), 2003
- *The Design of the Borealis Stream Processing Engine*, Abadi et. al., Proc. CIDR ’05, 2005
- *“One Size Fits All”: An Idea Whose Time Has Come and Gone*, Stonebraker and Cetentimel, Proc. ICDE ’05, 2005
- *Fault-tolerance in the Borealis distributed stream processing system*, Balazinska et. al., Proc. SIGMOD ’05, 2005
- *Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing*, Tatbul et. al., Proc. VLDB ’07, 2007