Data Streams

From Niagara Falls to Aurora Borealis

Cold stuff!

Examples of Data Stream Applications

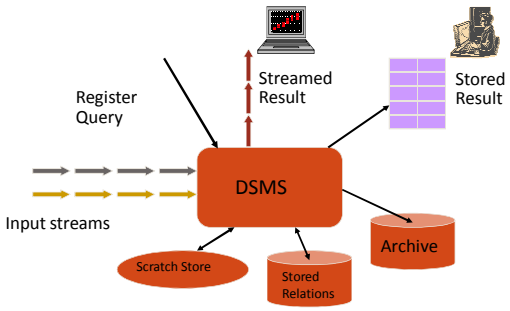
- Continuous, unbounded, rapid, time-varying streams of data elements (tuples).
  - Market Analysis
    - *Streams of Stock Exchange Data*
  - Critical Care
    - *Streams of Vital Sign Measurements*
  - Physical Plant Monitoring
    - *Streams of Environmental Readings*
  - Biological Population Tracking
    - *Streams of Positions from Individuals of a Species*
- DSMS = Data Stream Management System

DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- Access plan determined by query processor and physical DB design
- Transient streams (and persistent relations)
- Continuous queries
- Sequential access
- Unpredictable data characteristics and arrival patterns

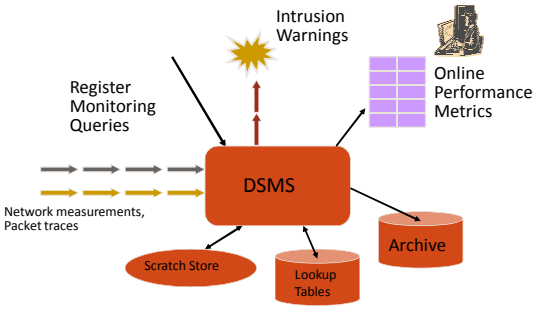
stanfordstreamdatamanager

The (Simplified) Big Picture



stanfordstreamdatamanager

(Simplified) Network Monitoring



stanfordstreamdatamanager

	Monitoring Apps	Traditional DBMS
Typical model	*Data Active Human Passive*	*Data Passive Human Active*
Managing History of values	*required*	*Very hard or inefficient*
Approximate query result	*required*	*Not supported*
Triggers & Alerts	*High Priority*	*Low Priority*
Real-time requirement	*required*	*Not supported*

Discussion 1

"Existing DBMS systems are ill suited for such applications since they target business applications." Do you think implementing monitoring systems using DBMSs is reasonable?

- If yes
  - How are traditional systems and monitoring systems similar?
  - Think of works and researches happened in DBMSs before, that Aurora benefits from or inspired by?
- If No
  - Which of those five assumptions is more problematic than others?
    - 1. DBMSs have a HADP model
    - 2. Current state of the data is the only thing that is important
    - 3. Triggers and alerts are second-class citizens
    - 4. Data elements are synchronized and that queries have exact answers
    - 5. No real-time services
  - Think of alternative architectures or models that can be used for monitoring applications?

Continuous Queries

- One time queries – Run once to completion over the current data set.
- Continuous queries – Issue once and continuously evaluate over a changing data set.
  - Example:
    - Notify me when the temperature drops below 30 deg. F
    - Notify me when prices of stock XYZ > $300
  - Popular paradigm among the users of Internet (has large amounts of frequently changing information)
    - Allow users to receive new results when available without having to issue same query repeatedly.
    - Need to support millions of queries to scale to the Internet.

Discussion 2

- What are some of the challenges in building continues query processors for temporal and/or spatio-temporal data streams?

* Some of the examples of spatio-temporal applications are E911, traffic monitoring, and location aware services dealing with moving objects

NiagaraCQ: A Scalable Continuous Query System for Internet Databases

Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang
Computer Sciences Department
University of Wisconsin-Madison
Modified slides by those from Le Cuong

What's NiagaraCQ?

- A distributed database system for querying distributed XML data sets using a query language like XML-QL.
- Employs
  - Query Grouping
    - Groups allow the "common parts" of two or more queries to be shared.
    - Each individual query in a query group shares the results from the execution of the "group plan".
  - Caching to boost performance.
  - Pull and Push models for detecting heterogeneous data source changes.
- Supports both change based and timer based CQs.

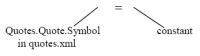
Basics - Expression Signature

- Represent the same syntax structure, but possibly different constant values, in different queries.
- Expression signatures allow queries with the same syntactic structure to be grouped together to share computation

```
Where <Quotes> <Quote>
        <Symbol>INTC</>
      </> </> element_as $g
in "http://www.cs.wisc.edu/db/quotes.xml"
construct $g
```

```
Where <Quotes> <Quote>
        <Symbol>MSFT</>
      </> </> element_as $g
in "http://www.cs.wisc.edu/db/quotes.xml"
construct $g
```

XML-QL examples (Fig. 3.1)



Expression Signature (Fig. 3.2)

Basics – Group Plan

- The query plan shared by all queries in the group. It is derived from the common part of all single query plans in the group.

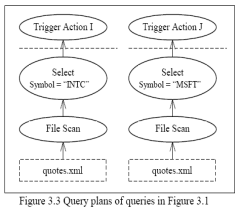


Figure 3.3 Query plans of queries in Figure 3.1

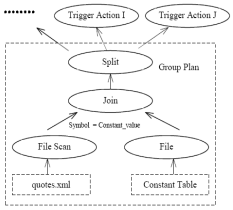


Figure 3.5 Group plan for queries in Figure 3.1

NiagaraCQ Novelty

- Incremental group optimization strategy.
- Incremental evaluation of continuous queries.

Query Grouping

- Groups are created for queries based on their expression signatures. Consists of 3 parts:
  - *Group signature*: The common expression signature of all queries in the group.
  - *Group constant table*: The *group constant table* contains the signature constants of all queries in the group.
  - *Group Plan*

Constant value	Destination buffer
….	….
INTC	Dest. i
MSFT	Dest. j
….	….

Figure 3.4 an example of group constant table

Incremental Grouping Algorithm

- When a new query is submitted:
  1. Group optimizer traverses query plan bottom up to match its expression signature with the signatures of existing groups.
  2. Group optimizer breaks new query plan into two parts
  3. Lower part of query is removed. Upper part of query is added onto the group plan.
  4. If constant table does not have an entry "AOL", it will be added and a new destination buffer allocated.
- If no match, a new group will be generated for this signature and added to the group table.

```
Where <Quotes>
<Quote>
  <Symbol>AOL</>
  </></>
  element_as $g
in
"http://www.cs.wisc.edu/
db/quotes.xml"
construct $g
```

Figure 3.6 XML-QL query examples

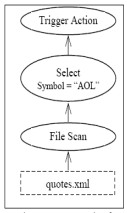
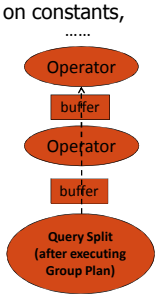


Figure 3.7 Query plan for query in Figure 3.6

Buffering output of the Split Operator

- After executing the Group Plan, the query-split operator produces outputs based on constants, subsequent group plans, etc.
- How do we buffer the o/p ?
  - Pipelined scheme
  - Intermediate Files



Pipeline approach

- Tuples are pipelined from the output of one operator into the input of the next operator.
  - Doesn't work for grouping timer-based CQ's. It's difficult for a split operator to determine which tuple should be stored and how long they should be stored for.
  - The combine plan may be very large requires resources beyond the limits of system.
  - A large portion of the query plan may not need to be executed at each query invocation.
  - One query may block many other queries.

Materialized Intermediate Files

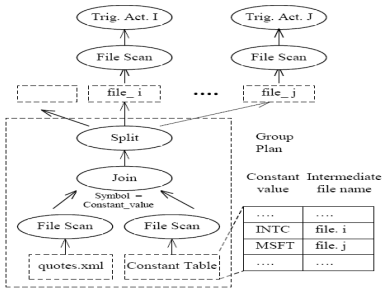


Figure 3.8 query-split scheme using intermediate files

Materialized Intermediate Files (cont.)

- **Advantages**
  - Intermediate files and data sources are monitored uniformly.
  - Each query is scheduled independently.
  - The potential bottleneck problem of the pipelined approach is avoided.
- **Disadvantages**
  - Extra disk I/Os.
  - Split operator becomes a blocking operator.

Timer-based Continuous Queries

- Grouped in the same way as change-based queries except that the time information needs to be recorded at registration time.
- Challenges
  - Monitoring the timer events (determining whether data has changed, pull new data?).
  - Sharing the common computation becomes difficult due to the various time intervals.
- Timer-based continuous queries fires at specific times, but only if the corresponding input files have been modified.

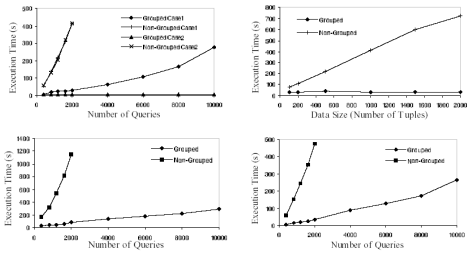
NiagaraCQ Novelty

- Incremental group optimization strategy.
- Incremental evaluation of continuous queries.

Incremental CQ Evaluation

- Incremental evaluation allows queries to be invoked only on the changed data.
- For each file, on which CQ's are defined, NiagaraCQ keeps a "delta file" that contains recent changes.
- Queries are run over the delta files whenever possible instead of their original files.
- A time stamp is added to each tuple in the delta file. NiagaraCQ fetches only tuples that were added to the delta file since the query's last firing time.

Some performance comparisons



Conclusion

- Incremental grouping methodology makes group optimization more scalable.
- The query-split scheme requires minimal changes to a general purposed query engine. In this model, both timer-based and change-based continuous queries can be grouped together for event detection and grouped execution.
- Incremental evaluation of continuous queries, use of both pull and push models for detecting heterogeneous data source changes and a caching mechanism further improve scalability.

Discussion 3

- Is this approach feasible?
- Is this a better application for XML or for relational data?
- Justify your answer and provide examples/scenarios where you would use a system like NiagaraCQ.

Aurora

Don Carney	Brown University
Uğur Çetintemel	Brown University
Mitch Cherniack	Brandeis University
Christian Convey	Brown University
Sangdon Lee	Brown University
Greg Seidman	Brown University
Michael Stonebraker	MIT
Nesime Tatbul	Brown University
Stan Zdonik	Brown University

Slides borrowed from http://web.cs.wpi.edu/~cs561/s07/lectures/STREAM/aurora-cs561.ppt

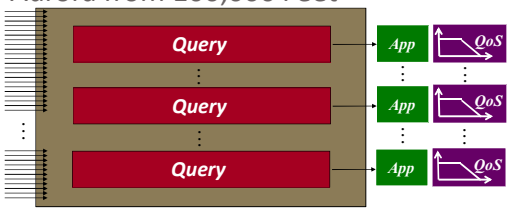
Background

- MIT/Brown/Brandeis team
- First Aurora, then Borealis
  - Practical system
  - *Designed for Scalablility: 10^6 stream inputs, queries*
  - *QoS-Driven Resource Management*
  - *Stream Storage Management*
  - *Realiability/ Fault Tolerance*
  - *Distribution and Adaptivity*
- First stream startup: StreamBase
  - Financial applications

Outline

1. Aurora Overview/ Query Model
2. Runtime Operation
3. Adaptivity

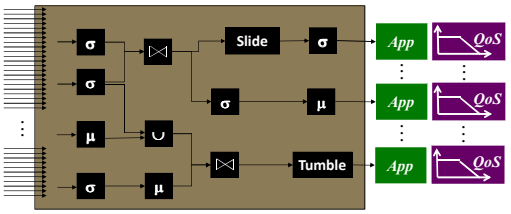
Aurora from 100,000 Feet



Each Application Provides:

- *A Query over input data streams*
- *A Quality-Of-Service Specification (QoS) (specifies utility of partial or late results)*

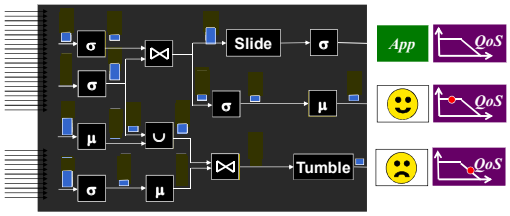
Aurora from 100 Feet



Query Operators (Boxes)

- *Simple:* *FILTER, MAP, RESTREAM*
- *Binary: UNION, JOIN, RESAMPLE*
- *Windowed:* *TUMBLE, SLIDE, XSECTION, WSORT*

Aurora in Action

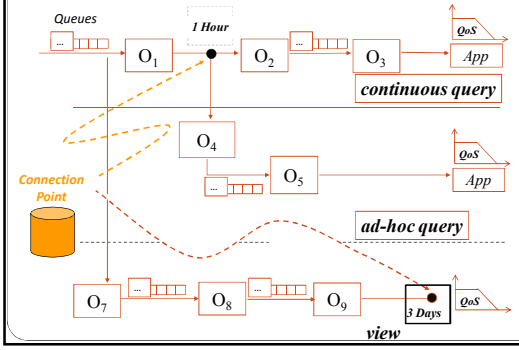


Arcs → Tuple Queues

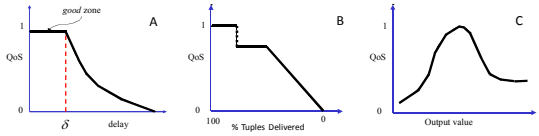
"Box-at-a-time" Scheduling

Outputs Monitored for QoS

Continuous and Historical Queries



Quality-of-Service (QoS)



Specifies "Utility" Of Imperfect Query Results

Delay-Based (specify utility of late results)

Delivery-Based, Value-Based (specify utility of partial results)

QoS Influences…

Scheduling, Storage Management, Load Shedding

Discussion 4

The authors state: "*Asking the application administrator to specify a multidimensional QoS function seems impractical. Instead, Aurora relies on a simpler tactic, which is much easier for humans to deal with: for each output stream, we expect the application administrator to give Aurora a two-dimensional QoS graph based on the processing delay of output tuples produced* " and "*the application administrator can give Aurora two additional QoS graphs for all outputs in an Aurora system. The first shows the percentage of tuples delivered.* " and *the second one is "The possible values produced as outputs appear on the horizontal axis, and the QoS graph indicates the importance of each one.*"

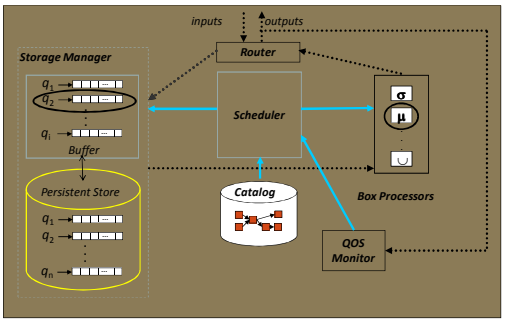
- Does this seem easier?
- Does it make sense to you?
- How could a good or bad graph affects the performance of Aurora? (as the Scheduler, Storage Manager and Load Shedding are dependant on the QoS function)

Outline

1. Aurora Overview
2. → Runtime Operation
3. Adaptivity

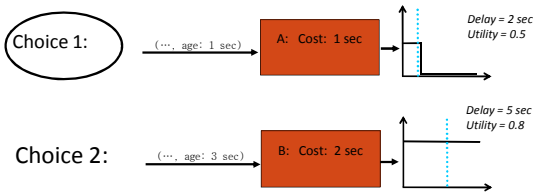
Runtime Operation

Basic Architecture



Runtime Operation

Scheduling: Maximize Overall QoS



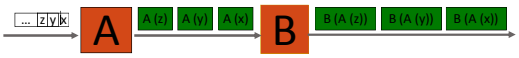
Schedule Box A now rather than later

Ideal: Maximize Overall Utility

Presently exploring scalable heuristics (e.g., feedback-based)

Runtime Operation

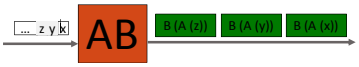
Scheduling: Minimizing Per Tuple Processing Overhead



Default Operation: *Context Switch*

Train Scheduling:

Box Trains:



Tuple Trains:

Runtime Operation

Storage Management

1. Run-time Queue Management

   Prefetch Queues Prior to Being Scheduled

   Drop Tuples from Queues to Improve QoS

2. Connection Point Management

   Support Efficient (Pull-Based) Access to Historical Data
   E.g., indexing, sorting, clustering, …

Outline

1. Aurora Overview
2. Runtime Operation
3. → Adaptivity

Query Optimization

Compile-time, Global Optimization Infeasible

Too Many Boxes
Too Much Volatility in Network, Data

Dynamic, Local Optimization

Threshold when to optimize

Adaptivity

Load Shedding

1. Two Load Shedding Techniques:
   - *Random Tuple Drops*

     Add DROP box to network (DROP a special case of FILTER)
     Position to affect queries w/ tolerant delivery-based QoS reqts
   - *Semantic Load Shedding*

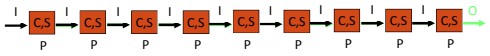
     FILTER values with low utility (acc to value-based QoS)
2. Triggered by QoS Monitor

   e.g., after Latency Analysis reveals certain applications are continuously receiving poor QoS

Adaptivity

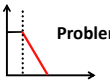
Detecting Overload

Throughput Analysis



Latency Analysis

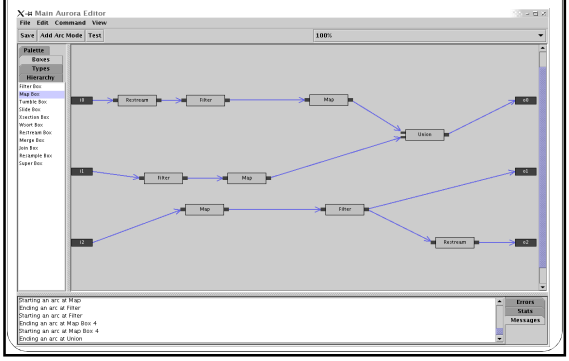
Monitor each application's Delay-based QoS



Problem: Too many apps in "bad zone"

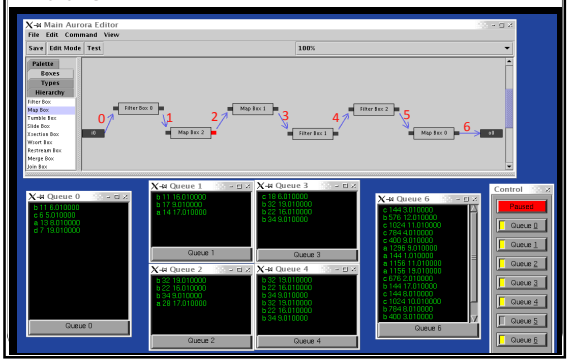
Implementation

GUI



Implementation

Runtime



Conclusions

Aurora Stream Query Processing System

1. *Designed for Scalability*
2. *QoS-Driven Resource Management*
3. *Continuous and Historical Queries*
4. *Stream Storage Management*
5. *Implemented Prototype*

Web site: ***www.cs.brown.edu/research/aurora/***

Aurora…

- Aurora is the Latin word for "dawn".
- A polar light (caused by solar wind and seen near the poles).
- The collective noun for a group of polar bears.
- Several aircraft.
- Several vessels.
- Several Companies.
- In space:
  - An asteroid, discovered by J. C. Watson, in september 6, 1867.
  - The Aurora Programme, a strategy of the European Space Agency.
- In fiction:
  - A superhero in the Marvel Universe.
  - One of the Spacer worlds in Isaac Asimov's fiction
- One of at least four distinct music groups: a UK house group, also known as Aurora UK; a California-based ambient group; a contemporary Christian R&B group; a Mexican Latin music band.
- The name of the game engine that runs *Neverwinter Nights*, the toolset is called the Aurora toolset because of this.
- AND the aurora system as presented today.

Courtesy: Qing Cao - CS@UVA

Merci ☺