Monitoring Streams : A New Class of Data Management Applications

CPSC 504: DATA MANAGEMENT
2009
PRESENTER: YONG
DISCUSSION : BRENDAN

Outline

- Motivation
  - -5 assumptions of traditional DBMS
  - -Monitoring applications
  - -Rethink the fundamental
- Aurora System Model
- Aurora Run-time architecture
  - QoS in Aurora
  - Real-time Scheduling
- Conclusion

5 assumptions of traditional DBMS

1. Passive repository: Human-Active, DBMS-Passive (HADP) model
2. The current of state of the data is important: Previous data needs to be extracted from the log
3. Triggers and alerts as second-class citizens
4. Perfect synchronization of data elements and exact query answers
5. No real-time services from applications

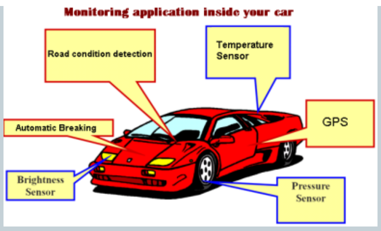
So what's wrong with this assumption?

Monitoring applications : are those where streams of information, triggers, real-time requirements, and imprecise data are prevalent.

So what's wrong with this assumption?

5 assumptions

1. HADP model
2. Only the current data is important
3. Triggers and alerts as second-class citizens
4. Perfect synchronization of data elements and complete data
5. No real-time services



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

So what's wrong with this assumption?

	Monitoring Application	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*
Trigger oriented	*required*	*Limited support*
Real-time requirement	*required*	*Not supported*

So what's wrong with this assumption?

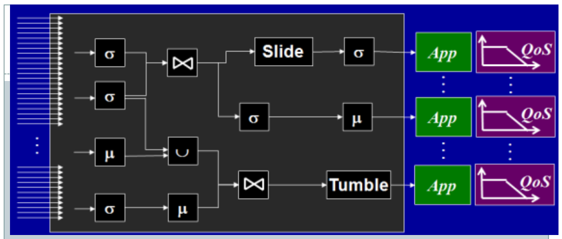
SO!

All 5 assumptions are problematic for motoring applications!

Aurora System Model

- So, the solution "*Aurora*", which is designed to better support monitoring applications

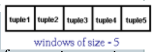
-Stream data
-Triggers
-Imprecise data
-Real-time requirement



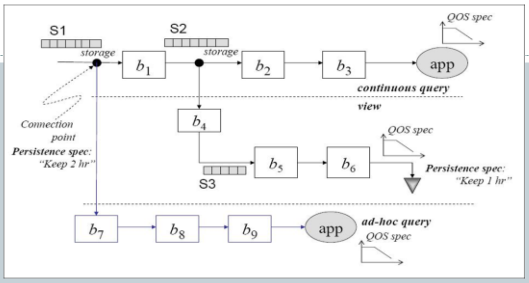
Aurora: process incoming streams in the way defined by an applications (data-flow system : Aurora Network)
Data sources (stream) : A stream in Aurora is a sequence of tuples from a given data source, and each tuple is time stamped upon entry to Aurora
Boxes : performs operations on incoming stream of data

Boxes : Operations

8 primitive operators (Box)

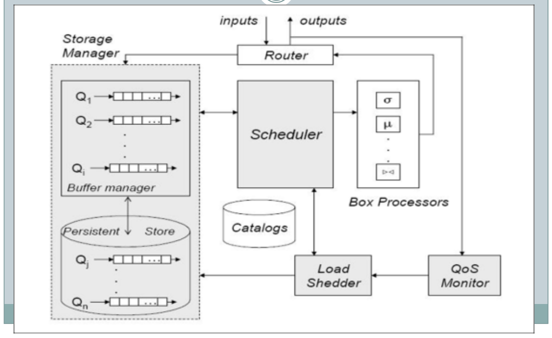


- Windowed : Operate on a set of consecutive tuples from a stream at a time. Applies function to a windows and advances the window to capture a new set of tuples.
  - Slide : advances a window by 'sliding' it downstream by some no of tuples.
  - Tumble: consecutive windows don't have overlap
  - Latch: maintain internal state between window.
  - Resample : produce synthetic stream.
- Non-windowed: single tuple at a time
  - Filter : condition
  - Map : apply a function to every tuple
  - GroupBy : partition incoming tuples across multiple streams to groups
  - Join : pairs tuples from input streams

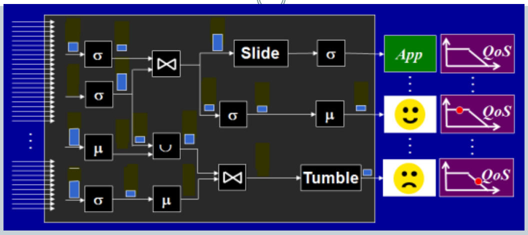


3 kinds of query supported
Continuous
View
Ad-Hoc Query

Aurora Run-time architecture



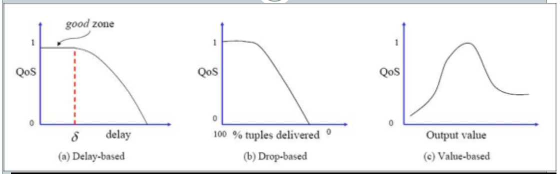
QoS: Quality of Service



Quality of Service (QoS) must be provided by the **application administrator!**

The QoS monitor constantly monitors system performance and activates load shedder (ex. Drop tuples) when it is needed, that is, the system performance is degrading by data overload.

QoS: Quality of Service



Discussion

- 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." Does this seem easier? Does it make sense to you?

Real Time Scheduling

- Scheduling decision on QoS is not enough!

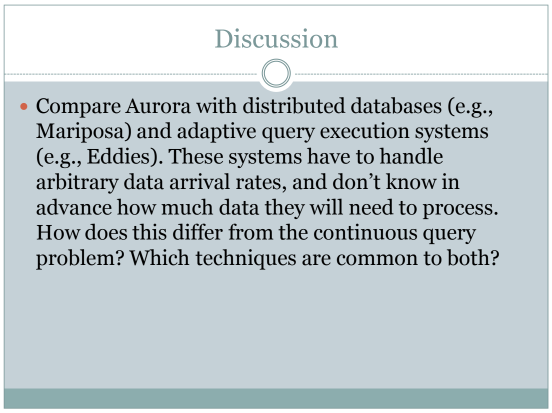
Maximize overall QoS + reduce overall end to end tuple execution costs!

But how?

The objective is to not only maximize overall QoS but also reduce overall tuple execution costs

Conclusion

- **Aurora Stream Query Processing System**
- *Designed for Scalability*
- *QoS-Driven Resource Management*
- *Continuous and Historical Queries*
- *Stream Storage Management*
- ***Implemented Prototype***
  www.cs.brown.edu/research/aurora/



Discussion

- Compare Aurora with distributed databases (e.g., Mariposa) and adaptive query execution systems (e.g., Eddies). These systems have to handle arbitrary data arrival rates, and don't know in advance how much data they will need to process. How does this differ from the continuous query problem? Which techniques are common to both?