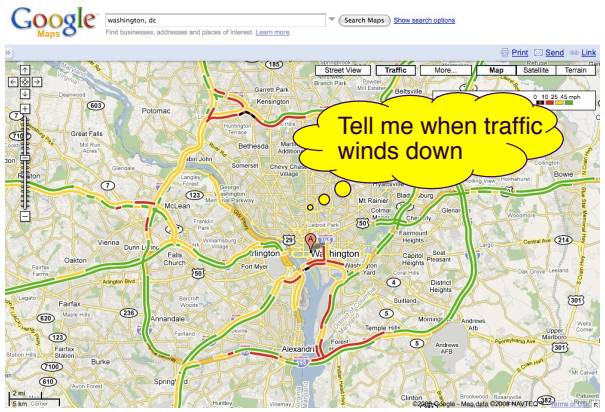
Data Stream Management Systems

(An In-Memory System)

Monitoring Applications - Financial

Monitoring Applications - Traffic



Monitoring Applications - Environment

Employees at the control center of the Environmental Protection Monitoring Service in Beijing monitor pollution levels, 17 August 2007.

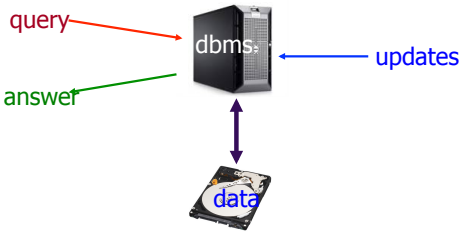
Monitoring Applications - Web

- Travel Alerts (TripCase and many apps)
  - Sends free updates to cell phone
    - Flight delays, gate changes, baggage claim, stormy weather
- UV Alerts
  - System integrates (1) customer's location information with (2) ultraviolet radiation data from sensors
  - System alerts you to reapply sunscreen based on:
    - location, skin tone, and type of lotion!

Why are monitoring apps different?

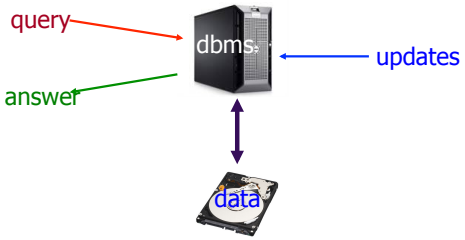
- Data needs to be process continuously as it arrive
- Queries execute continuously
- Response time / latency in seconds ("real-time")

Queries in Traditional DBMS



Store-then-query processing

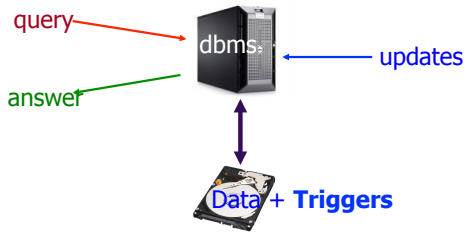
Continuous Queries in Traditional DBMS



Store-then-query-and-query-again processing

Continuous Queries in Active DBMS

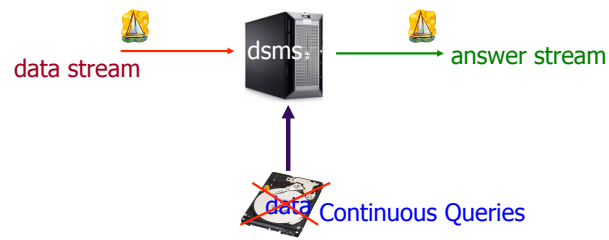
circa 1990-2000



Store-store-then-trigger-processing

Data Stream Management Systems

circa 2001-2010



Conveyor-belt data processing

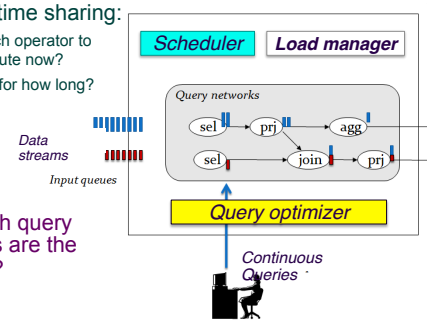
DSMS = Conveyor Belt Model

AQSIOS

- Our State of the art query processing system that efficiently utilizes resource

CPU time sharing:

- Which operator to execute now?
- And for how long?



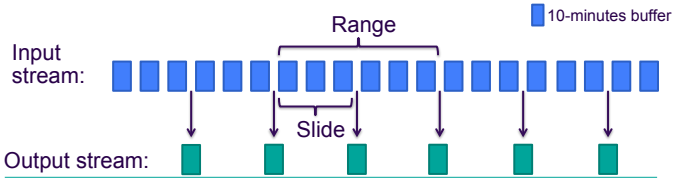
What if the system is *overloaded?*

- *Shed data to meet the near-real-time requirement*

Which query plans are the best?

Window Operators

- Window operator periodically produces sets of tuples
- Properties of *sliding windows*
  - **Range**: width of the window
  - **Slide**: How often to emit new sets of tuples (time, count)
- Aggregate CQ: *Every 30 minutes, return the average stock price over the last hour*
  - Range: 1 hour; Slide: 30 minutes



Aggregate Continuous Queries (ACQs)

```
SELECT    <Aggregation(attribute)>
FROM      <named_stream_list>
WHERE     <Predicate(attribute-list)>
GROUP BY  <attribute list>
```

Range = <user preference> and **Slide** = <user preference>

```
SELECT AVG(trade) FROM stock_stream
Range= 1 hour and Slide= 10 mins
```

```
SELECT AVG(trade) FROM stock_stream
Range= 10 minutes and Slide= 1 min
```

```
SELECT AVG(trade) FROM stock_stream
Range= 8 hours and Slide= 1 hour
```

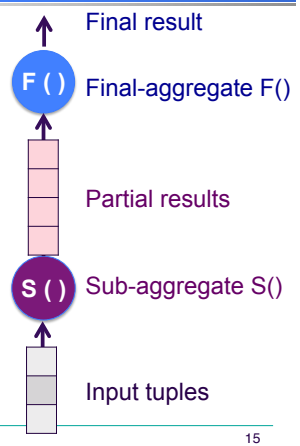
How to optimize the processing of **multiple** ACQs?

Processing Aggregate Continuous Queries

Naïve Implementation:
After each slide-interval:
Produce the aggregate result of a new window instance from **scratch** (i.e., raw tuples)

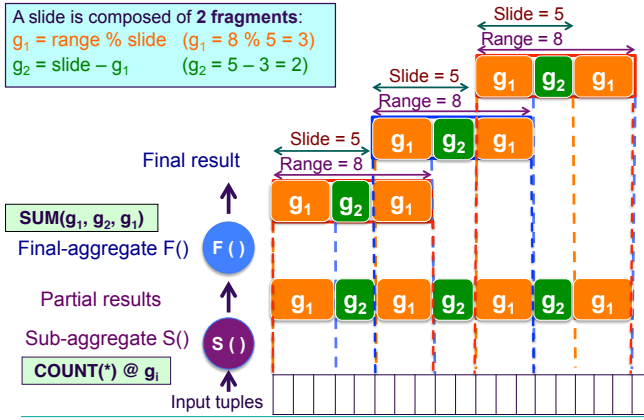
Partial Aggregation:
[Li et al., SIGMOD'05]

Paired Windows:
[Krishnamurthy et al., SIGMOD'06]



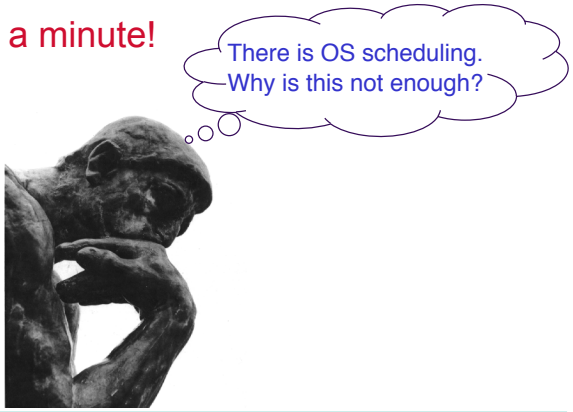
Paired Windows [Krishnamurthy et al., SIGMOD'06]

A slide is composed of **2 fragments**:
g_1 = range % slide (g_1 = 8 % 5 = 3)
g_2 = slide – g_1 (g_2 = 5 – 3 = 2)

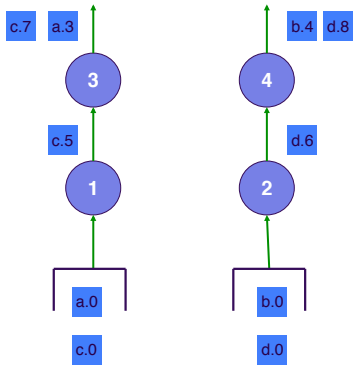


Continuous Query Scheduling

Wait a minute!

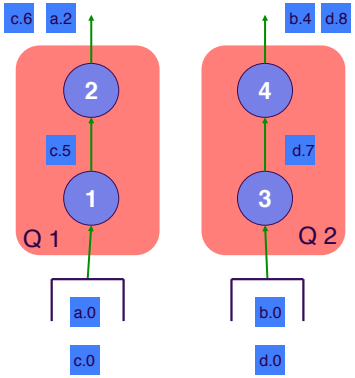


Round-Robin Scheduling (Operators)



- Each operator is considered to be a task

Round-Robin Scheduling (Queries)



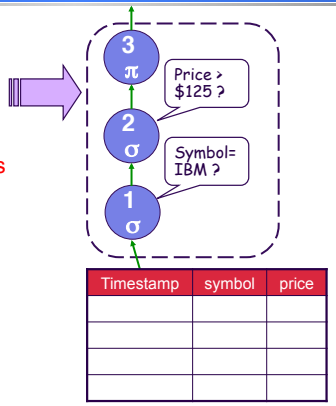
Two-level scheduling:

- Queries execute in round-robin
- Operators within queries are executed in pipelined fashion

Continuous Queries (CQ)

Example: Alert when IBM stock goes above $125

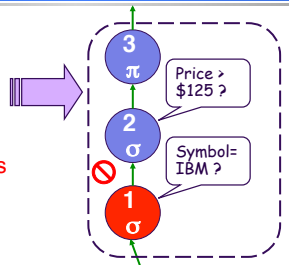
- CQ = a set of operators



Continuous Queries (CQ)

Example: Alert when IBM stock goes above $125

- CQ = a set of operators

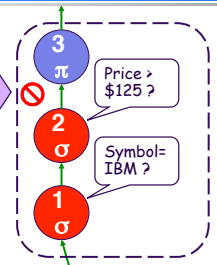


Timestamp	symbol	price
10:01	MSFT	$28

Continuous Queries (CQ)

Example: Alert when IBM stock goes above $125

- CQ = a set of operators



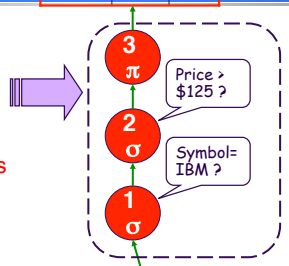
Timestamp	symbol	price
10:01	MSFT	$28
10:02	IBM	$110

CQs

10:07	IBM	$127

Example: Alert when IBM stock goes above $125

- CQ = a set of operators



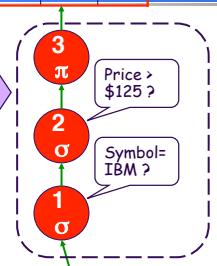
Timestamp	symbol	price
10:01	MSFT	$28
10:02	IBM	$90
10:07	IBM	$127

CQs

10:07	IBM	$127

Example: Alert when IBM stock goes above $125

- CQ = a set of operators
- Each operator is associated with 2 properties:
  - Operator Cost (*c*)
    - Processing time
  - Operator Selectivity (*s*)
    - Not all inputs are events!



Timestamp	symbol	price
10:01	MSFT	$28
10:02	IBM	$90
10:07	IBM	$127

Event-based QoS Metric: Response Time

- The interval between an event's arrival and departure
  - Example: Time between IBM=$125 and notification
  - Metric: Response Time (RT), Delay, Latency, …
  - Discarded tuples do not contribute to the metric
- Average Response Time for N events = $\frac{1}{N}\sum_{i=1}^{N} RT_i$
- Response Time = processing time + queuing delay

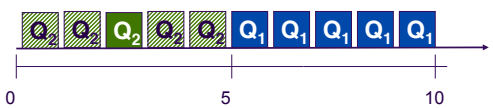
Example: Impact of Selectivity

	Selectivity	Cost
Q1	1.0	1 ms
Q2	0.2	1 ms

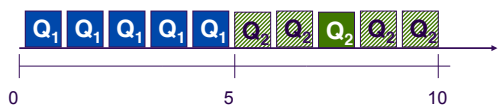
	RT
Q_2 then Q_1	7.1
Q_1 then Q_2	3.8

- Input: 5 pending tuples arrived at time 0

Q_2 then Q_1



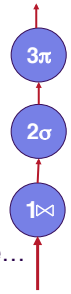
Q_1 then Q_2



Highest Rate Policy

- "Shortest time to detect an event" = "highest output rate"
- Assign each query a priority equals to its output rate

$$OR_i = \frac{S_i}{C_i^{avg}}$$



- At each scheduling point:

  schedule the query with the highest global output rate…

  Highest Rate Policy (HR)

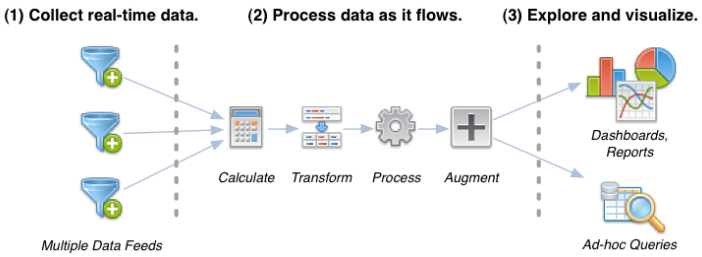
  [Sharaf et. al., VLDB' 06, ACM TODS08]

Summary

- **DBMS** = Database Management System
  - Data are stored
  - Queries come in
  - Answers come out, based on current state of database
  - Useful when updates come in batches and are not many
- **DSMS** = Data Stream Management System
  - Queries are stored (registered, ahead of time)
  - Data come in continuously, as a stream
  - Answers come out continuously, as a stream
  - Useful for monitoring applications

Continuous / Real-time Analytics

circa 2011- now



DSMS Challenges

- Given:
  - Continuous data streams [Variety]
  - High volume of updates [Volume]
  - Bursty data arrival [Variability]
  - Near real-time processing [Velocity]
  - Unreliable sources & connections [Veracity]
  - Protect privacy & provide security [Visibility]
  - Explore & visualize [Value]
- How to efficiently handle this …. Big Data?

How to handle the DSMS Challenges

- Query Optimization
  - new architectures, distribution/Cloud
  - New operator implementations
- Load Balancing
  - scale-out, query & operator migration,
  - shedding (drop tuples to meet response time)
- Scheduling of Continuous Queries
  - QoS
- Privacy and Security
- What about a "Store" operator ?
  - consistency, reliability/fault tolerance

AstrapiDB: Real-time Analytics