Data Stream Query Processing

Nikos Koudas University of Toronto
and
Divesh Srivastava AT&T Labs-Research

Stream Map

- Part I: Motivation (45 min)
  - Data streams: what, why now, applications
  - Data streams: architecture and issues
- Part II: Query processing (90 min)
- Part III: XML, open issues (45 min)

Data Streams: What and Where?

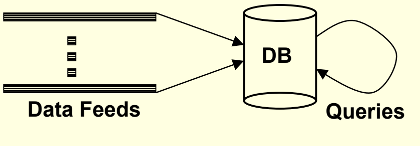
- A **data stream** is a (potentially unbounded) sequence of tuples

- **Transactional** data streams: log interactions between entities
  - Credit card: purchases by consumers from merchants
  - Telecommunications: phone calls by callers to dialed parties
  - Web: accesses by clients of resources at servers

- **Measurement** data streams: monitor evolution of entity states
  - IP network: traffic at router interfaces
  - Sensor networks: physical phenomena, road traffic
  - Earth climate: temperature, moisture at weather stations

Data Streams: Why Now?

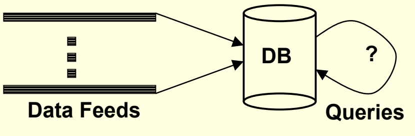
- Haven't data feeds to databases always existed? Yes
  - Modify underlying databases, data warehouses
  - Complex queries are specified over stored data



- Two recent developments: application- and technology-driven
  - Need for sophisticated near-real time queries/analyses
  - Massive data volumes of transactions and measurements

Data Streams: Real-Time Queries

- With traditional data feeds
  - Simple queries (e.g., value lookup) needed in real-time
  - Complex queries (e.g., trend analyses) performed offline
- Now need sophisticated near-real time queries/analyses
  - AT&T: fraud detection on call detail tuple streams
  - NOAA: tornado detection using weather radar data



Telecommunications Application: Fraud Detection

- Business Challenge: AT&T wanted to track calling pattern of each of ~100M callers, and raise real-time fraud alerts
- Previous Approach: Handwritten, optimized C code, computing evolving **signatures** for each customer, looking for variations
- Issues: Signature computation is I/O intensive, often modified
- Solution: Using Hancock domain-specific language
  - Abstract logical/physical streams and signatures
  - Express I/O and CPU efficient signature programs cleanly
- Lesson: Essential to consider I/O issues for data streams

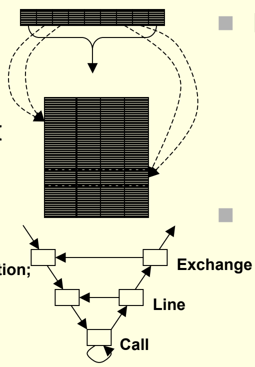
Hancock: Data Streams

```
typedef struct {
    line_t origin;
    line_t dialed;
    date_t connectTime;
    time_t duration;
    char isIncomplete;
    char isIntl;
    char isTollFree;
    …
} callRec_t;
```

- Physical data representation of tuples on disk
  - highly encoded structure
- Logical data representation
  - C struct
- Conversion functions
  - specified in Hancock

Hancock: Signature Programs

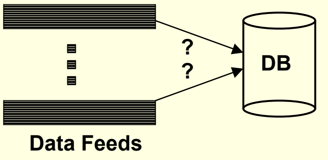
```
iterate (over calls
  sortedby origin
  filteredby noIncomplete
  withevents originDetect){

 event line_begin(lpn_t pn){
  cumSec.outTF = 0;
 }

 event call(callRec_t c){
  if (c.isTollFreeCall)
    cumSec.outTF += c.duration;
 }

 event line_end(lpn_t pn){
   mySig us = data<:pn:>;
   us.outTF = blend(cumSec.outTF, us.outTF);
   data<:pn:> := us;
 }}
```



- Hancock program paradigm:
  - stream-in, relation-out
  - block processing of data
  - multiple passes on block
- Hancock programs support:
  - iterating on sorted data
  - filtering
  - event clause hierarchy
- User-defined aggregation

Data Streams: Massive Volumes

- Now able to deploy transactional data observation points
  - AT&T long-distance: ~300M call tuples/day
  - AT&T IP backbone: ~50B IP flows/day
- Now able to generate automated, highly detailed measurements
  - NOAA: satellite-based measurement of earth geodetics
  - Sensor networks: huge number of measurement points



IP Network Application: Hidden P2P Traffic Detection

- Business Challenge: AT&T IP customer wanted to accurately monitor peer-to-peer (P2P) traffic evolution within its network
- Previous Approach: Determine P2P traffic volumes using TCP port number found in Netflow data
- Issues: P2P traffic might not use known P2P port numbers
- Solution: Using Gigascope SQL-based DSMS
  - Search for P2P related keywords within each TCP datagram
  - Identified 3 times more traffic as P2P than using Netflow
- Lesson: Essential to query massive volume data streams

IP Network Application: Web Client Performance Monitoring

- Business Challenge: AT&T IP customer wanted to monitor latency observed by clients to find performance problems
- Previous Approach: Measure latency at “active clients” that establish network connections with servers
- Issues: Use of “active clients” is not very representative

- Solution: Using Gigascope SQL-based DSMS
  - Track TCP synchronization and acknowledgement packets
  - Report round trip time statistics: latency

- Lesson: Essential to correlate multiple data streams

Gigascope: Features and Functions

- Gigascope is a fast, flexible data stream management system
  - High performance at speeds up to OC48 (2 * 2.48 Gbit/sec)

- Developed at AT&T Labs-Research
  - Collaboration between database and networking research

- Current libraries include
  - Traffic matrix by site or autonomous system
  - Detection of hidden P2P traffic
  - End-to-end TCP performance monitoring
  - Detailed custom performance statistics

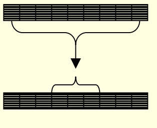
Gigascope: Data Streams

```
PROTOCOL IP (Layer2) {
    uint ipversion
}
PROTOCOL IPv4(IP) {
    uint hdr_length;
    uint service_type;
    uint total_length;
    uint id;
    bool do_not_fragment;
    bool more_fragments;
    uint offset;
    uint ttl;
    uint protocol;
}
```

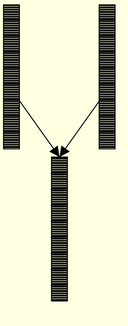
- GSQL queries get raw data from low level schemas
  - defined at packet level
  - inherits from lower layer
- Current schemas include
  - layer 2: ETH/HDLC
  - layer 3: IP/IPv4
  - layer 4: UDP/TCP/ICMP
  - layers 5-7: packet data

Gigascope: GSQL Queries

```
select tb, srcIP, sum(len)
from IPv4
where protocol = 6
group by time/60 as tb, srcIP
having count(*) > 5
```



```
select S.tstmp,
    S.srcIP, S.destIP,
    S.srcPort, S.destPort
    (A.tstmp – S.tstmp) as rtt
from tcp_syn S, tcp_syn_ack A
where S.srcIP = A.destIP
    and S.destIP = A.srcIP
    and S.srcPort = A.destPort
    and S.destPort = A.srcPort
    and S.tb = A.tb
```



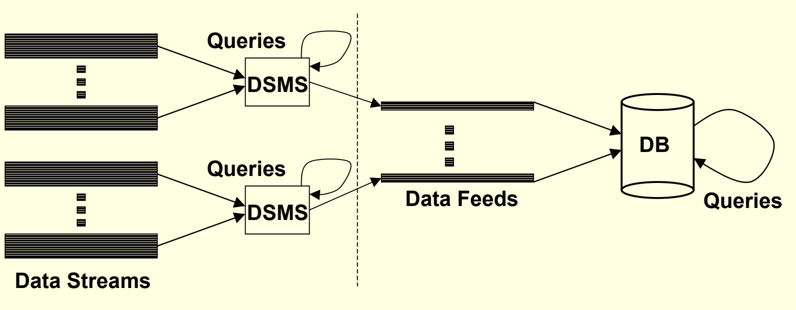
- GSQL queries support:
  - filtering, aggregation
  - merging data streams
  - joining data streams
  - hand-coded views
  - external functions
- GSQL query paradigm:
  - streams-in, stream-out
  - permits composability
  - permits data reduction

Stream Map

- Part I: Motivation (45 min)
  - Data streams: what, why now, applications
  - Data streams: architecture and issues
- Part II: Query processing (90 min)
- Part III: XML, open issues (45 min)

DSMS + DBMS: Architecture

- Data stream management system at multiple observation points
  - (Voluminous) streams-in, (data reduced) streams-out
- Database management system
  - Outputs of DSMS can be treated as data feeds to database



DSMS + DBMS: Architecture

Data Stream Systems

- Resource (memory, per-tuple computation) limited
- Reasonably complex, near real time, query processing
- Useful to identify what data to populate in database

Database Systems

- Resource (memory, disk, per-tuple computation) rich
- Extremely sophisticated query processing, analyses
- Useful to audit query results of data stream system

Databases vs Data Streams: Issues

Database Systems

- Model: persistent relations
- Relation: tuple set/bag
- Data Update: modifications
- Query: transient
- Query Answer: exact
- Query Evaluation: arbitrary
- Query Plan: fixed

Data Stream Systems

- Model: transient relations
- Relation: tuple sequence
- Data Update: appends
- Query: persistent
- Query Answer: approximate
- Query Evaluation: one pass
- Query Plan: adaptive

Really a continuum …

Relation: Tuple Set or Sequence?

- Traditional relation = set/bag of tuples
- Tuple sequences have been studied:
  - Temporal databases [TCG+93]: multiple time orderings
  - Sequence databases [SLR94]: integer “position” -> tuple
- Data stream systems:
  - Ordering domains: Gigascope [CJSS03], Hancock [CFP+00]
  - Position ordering: Aurora [CCC+02], STREAM [MWA+03]

Update: Modifications or Appends?

- Traditional relational updates: arbitrary data modifications
- Append-only relations have been studied:
  - Tapestry [TGNO92]: emails and news articles
  - Chronicle data model [JMS95]: transactional data
- Data stream systems:
  - Streams-in, stream-out: Aurora, Gigascope, STREAM
  - Stream-in, relation-out: Hancock

Query: Transient or Persistent?

- Traditional relational queries: one-time, transient
- Persistent/continuous queries have been studied:
  - Tapestry [TGNO92]: content-based email, news filtering
  - OpenCQ, NiagaraCQ [LPT99, CDTW00]: monitor web sites
  - Chronicle [JMS95]: incremental view maintenance
- Data stream systems:
  - Support persistent and transient queries

Query Answer: Exact or Approximate?

- Traditional relational queries: exact answer
- Approximate query answers have been studied [BDF+97]:
  - Synopsis construction: histograms, sampling, sketches
  - Approximating query answers: using synopsis structures
- Data stream systems:
  - Approximate joins: using windows to limit scope
  - Approximate aggregates: using synopsis structures

Query Evaluation: One Pass?

- Traditional relational query evaluation: arbitrary data access
- One/few pass algorithms have been studied:
  - Limited memory selection/sorting [MP80]: *n*-pass quantiles
  - Tertiary memory databases [SS96]: reordering execution
  - Complex aggregates [CR96]: bounding number of passes
- Data stream systems:
  - Per-element processing: single pass to reduce drops
  - Block processing: multiple passes to optimize I/O cost

Query Plan: Fixed or Adaptive?

- Traditional relational query plans: optimized at beginning
- Adaptive query plans have been studied:
  - Query scrambling [AFTU96]: wide-area data access
  - Eddies [AH00]: volatile, unpredictable environments
- Data stream systems:
  - Adaptive query operators
  - Adaptive plans

Data Stream Query Processing: Anything New?

Architecture

- Resource (memory, per-tuple computation) limited
- Reasonably complex, near real time, query processing

Issues

- Model: transient relations
- Relation: tuple sequence
- Data Update: appends
- Query: persistent
- Query Answer: approximate
- Query Evaluation: one pass
- Query Plan: adaptive

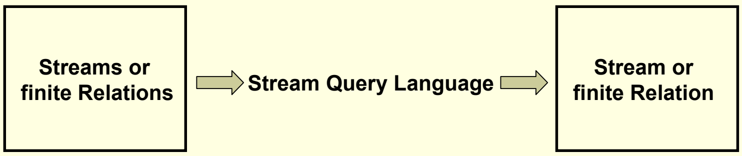
A lot of challenging problems ...

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
  - Stream query language issues (compositionality, windows)
  - Query operators
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: XML, open issues (45 min)

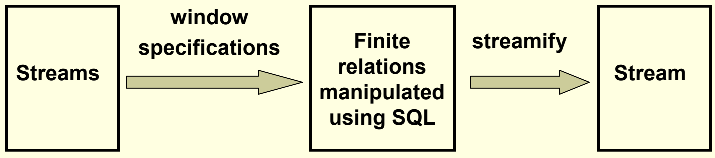
Stream Query Languages

- SQL-like proposals suitably extended for a stream environment:
  - Composable SQL operators
  - Queries reference/produce relations or streams
- GSQL [CJSS03]: SQL used by Gigascope
- CQL [ABW03]: SQL used by STREAM
- UDA-SQL [LWZ04]: Monotonic queries



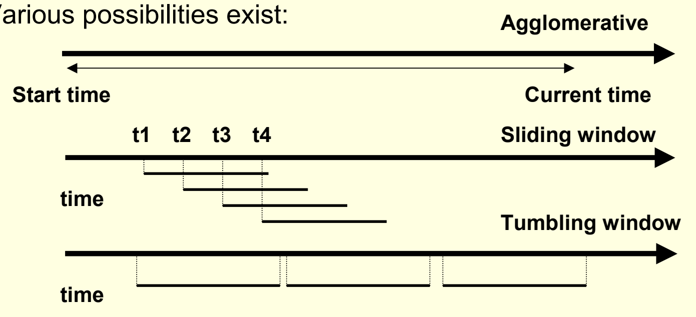
Windows

- Mechanism for extracting a finite relation from an infinite stream
- Various window proposals for restricting operator scope
  - Windows based on ordering attributes (e.g., time)
  - Windows based on tuple counts
  - Windows based on explicit markers (e.g., punctuations)
  - Variants (e.g., partitioning tuples in a window)



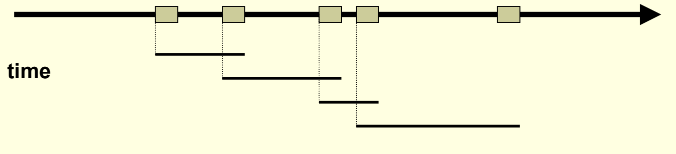
Ordering Attribute Based Windows

- Assumes the existence of an attribute that defines the order of stream elements/tuples (e.g., time)
- Let T be the window length (size) expressed in units of the ordering attribute (e.g., T may be a time window)
- Various possibilities exist:



Tuple Count Based Windows

- Window of size N tuples (sliding, tumbling) over the stream
- Problematic with non-unique time stamps associated with tuples
- Ties broken arbitrarily may lead to non deterministic output



Punctuation Based Windows [TMSF03]

- Application inserted “end-of-processing” markers
  - Each data item identifies “beginning-of-processing”
- Enables data item-dependent variable length windows
  - e.g., a stream of auctions
- Similar utility in query processing
  - Limit the scope of query operators relative to the stream

UDA-SQL [LWZ04]

- Key Idea: Only permit non-blocking queries on data streams
  - Non-blocking queries = monotonic queries
- Non-blocking RA cannot express all monotonic FO queries
  - Set difference (-) in RA is blocking wrt its second argument
  - Expression of “coalesce” and “until” use set difference
- Proposal: Support non-blocking user-defined aggregates
  - INITIALIZE, ITERATE: process tuples in an ordered fashion
  - NB-UDAs + Union = computable monotonic functions

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
  - Stream query language issues
  - Query operators (selections/projections, joins, aggregations)
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: XML, open issues (45 min)

Query Operators: Sample Stream

```
Traffic ( sourceIP      -- source IP address
          sourcePort    -- port number on source
          destIP        -- destination IP address
          destPort      -- port number on destination
          length        -- length in bytes
          time          -- time stamp
        );
```

Selections, Projections

- Selections, (duplicate preserving) projections are straightforward
  - Local, per-element operators
  - Duplicate eliminating projection is like grouping
- Projection needs to include ordering attribute [JMS95]
  - No restriction for position ordered streams

Select sourceIP, time
from Traffic
where length > 512

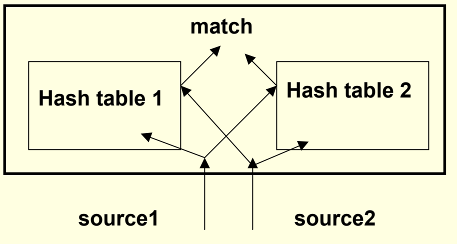
Join Operators

- General case of join operators problematic on streams
  - May need to join arbitrarily far apart stream tuples
  - Equijoin on stream ordering attributes is tractable [JMS95]
- Majority of work focuses on joins between streams with windows specified on each stream

Select A.sourceIP, B.sourceIP
from Traffic1 A [window T1], Traffic2 B [window T2]
where A.destIP = B.destIP

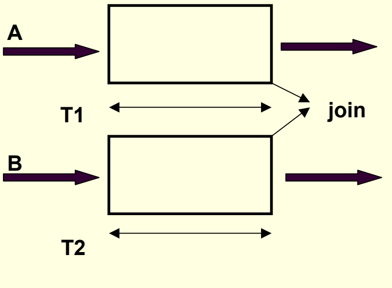
Join Operators: Background

- Symmetric Hash Joins [WA91]
  - Takes into account streaming nature of inputs



- XJoin [UF00]: extends Symmetric Hash Joins
  - overflowing inputs spilled to disk for later evaluation

Binary Joins [KNV03]



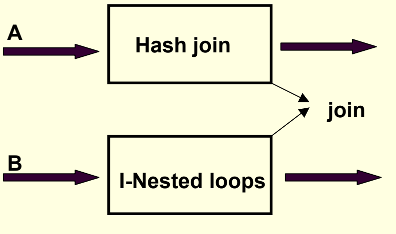
New A tuple:

- Scan B's window for joining tuples and output result
- Insert tuple into A's window
- **Invalidate** all expired tuples in A's window

Binary Joins: Issues

- How do existing join algorithms apply in this setting?
- Impact of stream arrival rate and resources in join processing
  - Introduce a cost model for each operator as a function of individual stream arrival rates (unit time based cost model)
  - Utilize the cost model to identify tradeoffs

Binary Joins: Key Observations



- Asymmetric join processing has advantages if arrival rates differ
- Goal: maximize tuple output
  - limited computational capability but sufficient memory
  - limited memory but sufficient computational capability

Joining Punctuated Streams [DMRH'04]

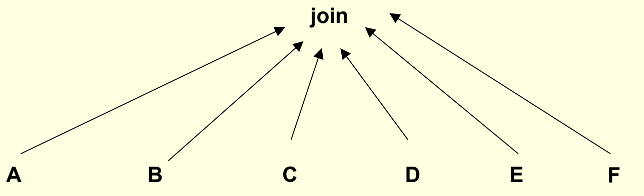
- Binary hash-based equi-join:
  - optimized for *reducing memory overhead*
  - optimized for *increasing data output rate*
- Fine-tunable execution logic: targeting various optimization goals
  - minimum memory overhead
  - maximum tuple output rate

Multi-way Joins

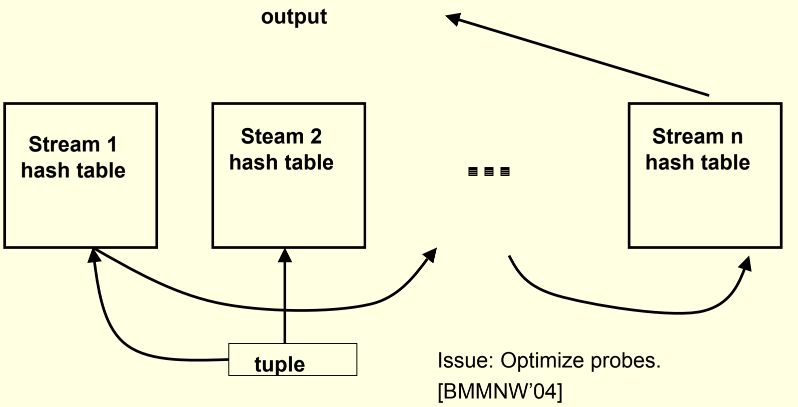
- Challenges during evaluation of n-way joins on streams:
  - evaluation order important
  - how to adapt traditional algorithms in this setting?
  - issues with varying arrival rates

Mjoin Operator [VNB03]

- Mjoin generalizes symmetric binary hash joins to work with multiple inputs
- Equijoins over attribute common to all streams
- Objective: maximize the output rate of the join operation

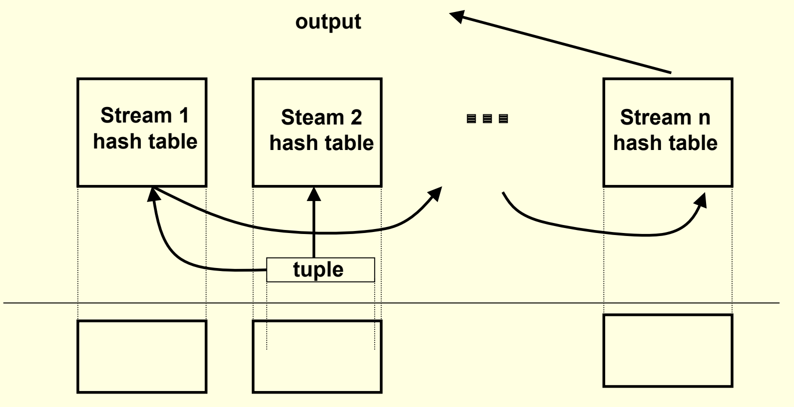


Mjoin: In-memory Operation



Issue: Optimize probes. [BMMNW'04]

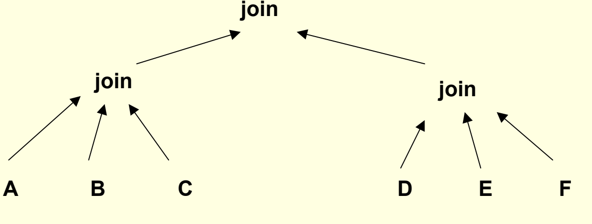
Mjoin: Disk to Memory



8/29/2005 U of Toronto 45

Mjoin: Observations

- As the number of input streams increases, the processing cost per input tuple increases
- In such cases, bushy plans of smaller (fewer input streams) Mjoin operators can be beneficial



8/29/2005 U of Toronto 46

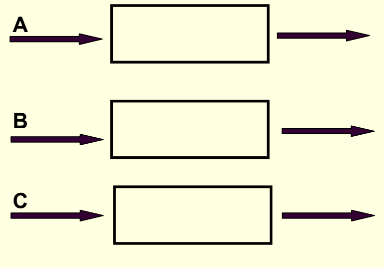
Multi-way Sliding Window Joins [GO03]

- Evaluation of n-way sliding window joins queries
  - n streams with associated sliding windows
  - continuously evaluate the joins of all n windows
- Two natural joins strategies
  - **eager**: join is evaluated each time a new tuple arrives in any of the input streams
  - **lazy**: join is evaluated with some pre-specified frequency, e.g., every t time units
- Relationship with incremental, bottom-up evaluation of recursive rules [BR87]

Eager Multi-way Joins

- Naïve approach extends the binary case
- Improved:
  - adapt join orders wrt origin of new tuple
  - Arrival on A
    - (A join B) join C
  - Arrival on B
    - (B join A) join C
  - Arrival on C
    - (C join A) join B

A join (B join C)



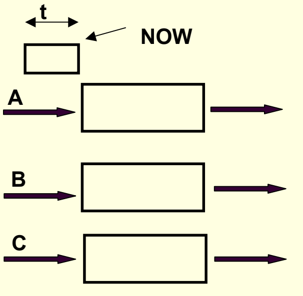
Nested Loops joins

Lazy multi-way joins

- Adapt eager multi-way joins in a lazy scenario
- Process all tuples that arrived since the last re-execution
- Imperative to apply lazy expiration for correctness

Insert new tuple into its window on arrival

At re-execution:
for each stream Si, for all k in Si
where NOW - t $\leq$ timestamp of k $\leq$ NOW
ComputeJoin(k, {S1, …, Si-1,Si+1,…,Sn})

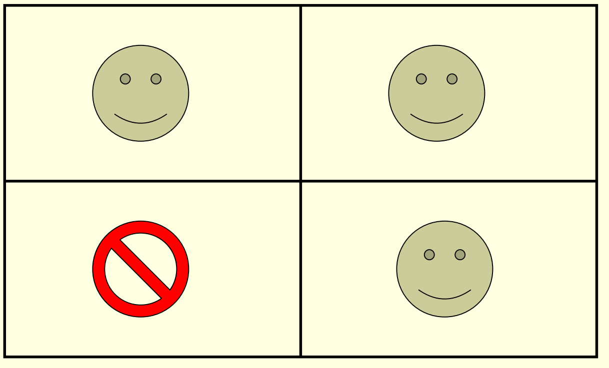


Multi-way joins

- Analytical model (using stream rates) to evaluate performance:
  - Eager algorithms better than the naïve approach
  - Improved lazy algorithms never worse than eager
- Similar algorithms can be instantiated for hash join strategies
  - Multi-way hash join algorithms more efficient than NL
  - More buckets for larger windows
  - Larger re-evaluation interval => larger hash tables

Strategies and Expirations

Eager tuple expiration | **Lazy tuple expiration**



Eager Evaluation

Lazy Evaluation

Aggregation

- General form:
  - **select** G, F1 **from** S **where** P **group by** G **having** F2 op ϑ
  - G: grouping attributes, F1,F2: aggregate expressions
- Aggregate expressions:
  - distributive: sum, count, min, max
  - algebraic: avg
  - holistic: count-distinct, median

Aggregation in Theory

- An aggregate query result can be streamed if group by attributes include the ordering attribute [JMS95]
- A single stream aggregate query “select G,F from S where P group by G” can be executed in bounded memory if [ABB+02]:
  - every attribute in G is bounded
  - no aggregate expression in F, executed on an unbounded attribute, is holistic
- Arasu et al. [ABB+02] derive conditions for bounded memory execution of aggregate queries on multiple streams

Aggregation in Bounded Memory

- Aggregate query execution not in bounded memory:

select length from Traffic [window T] where length > 512 group by length	≡	select distinct length from Traffic [window T] where length > 512

- Aggregate query execution in bounded memory:

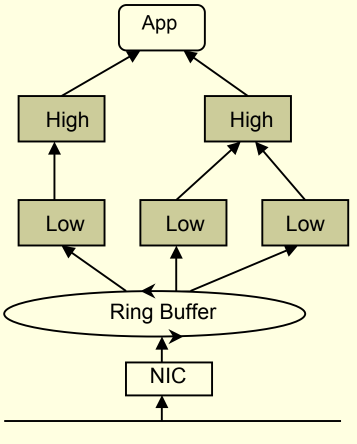
```
select length, count(*)
from Traffic [window T]
where length > 512 and length < 1024
group by length
```

Aggregation in Gigascope

- Grouping attributes contain window expressions restricting the scope of the group (e.g., temporally)
  - select peerid, tb, count(*) from Traffic group by time/60 as tb, f(destIP,'peerid.tbl') as peerid
  - time/60 is a minute-long shifting window
- Gigascope applies partial-aggregation on low-level data streams
  - bounded number of groups maintained at low level
  - unbounded number of groups maintainable at high level

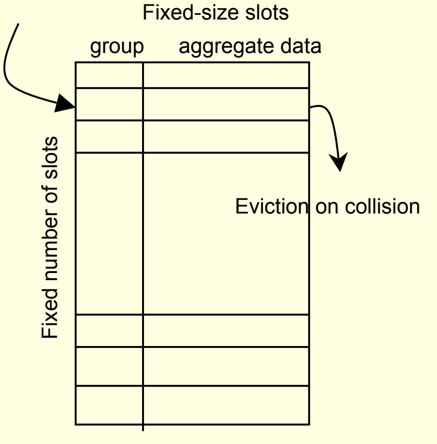
Gigascope: Run-Time Architecture

- Low-level queries perform fast selection, aggregation
- High-level queries complete complex aggregation



Gigascope: Low-Level Aggregation

- Fixed number of slots for groups, fixed size slot for each group
- Direct-mapped hashing
- Optimizations
  - Limited hash chaining reduces eviction rate
  - Slow eviction of groups when epoch changes



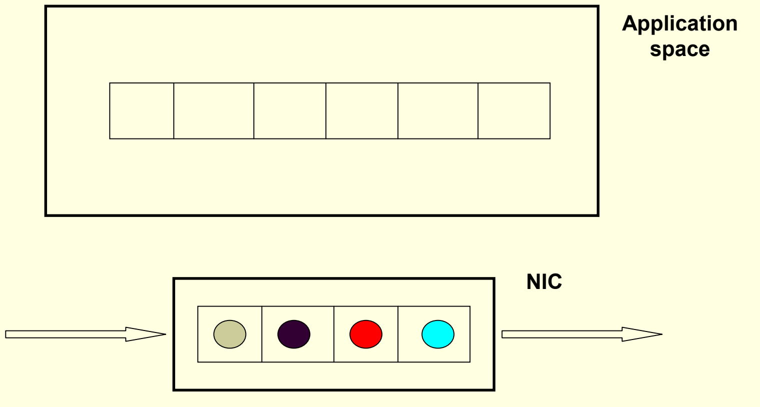
Gigascope: Query Splitting

select tb, srcIP, sum(len)
from IPv4
where protocol = 6
group by time/60 as tb, srcIP
having count(*) > 5

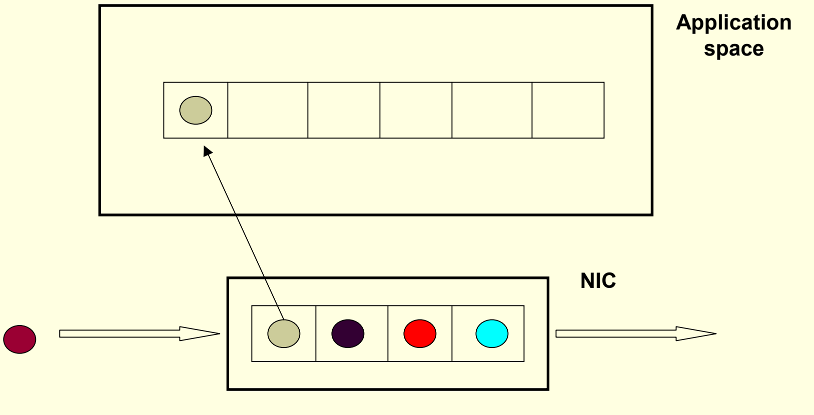
select tb, srcIP, sum(sumLen)
from SubQ
group by tb, srcIP
having sum(cnt) > 5

SubQ:
select tb, srcIP, sum(len) as
  sumLen, count(*) as cnt
from IPv4
where protocol = 6
group by time/60 as tb, srcIP

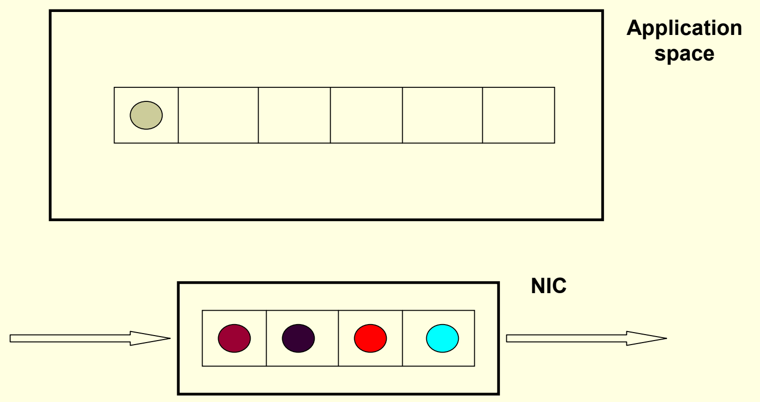
Aggregation in Gigascope



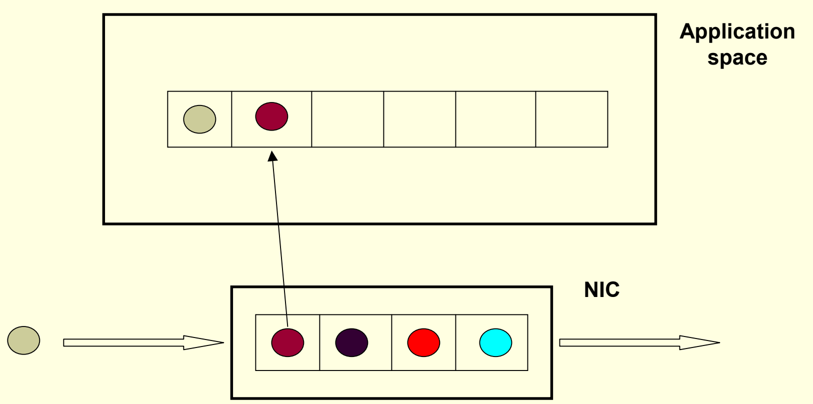
Aggregation in Gigascope



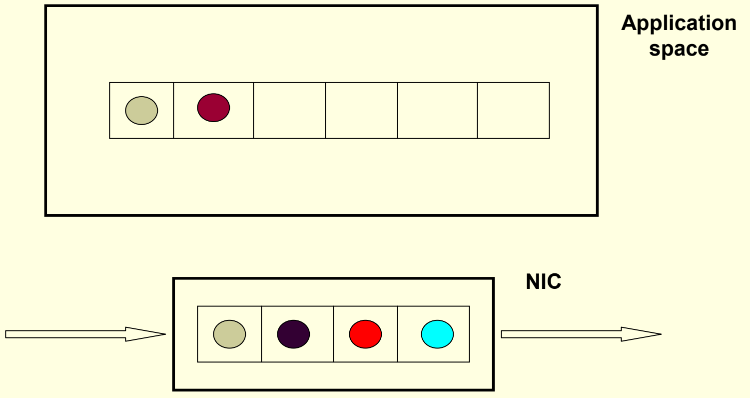
Aggregation in Gigascope



Aggregation in Gigascope



Aggregation in Gigascope



Aggregation & Approximation

- When aggregates cannot be computed exactly in limited storage, approximation may be possible and acceptable
- Examples:
  - select G, median(A) from S group by G
  - select G, count(distinct A) from S group by G
  - select G, count(*) from S group by G having count(*) > φ |S|
- Use summary structures: samples, histograms, sketches …
- Focus of different tutorial [GGR02]

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
  - Stream query language issues
  - Query operators
  - Optimization objectives (stream rate, resource limits, QoS)
  - Multi-query execution
  - Prototype systems
- Part III: XML, open issues (45 min)

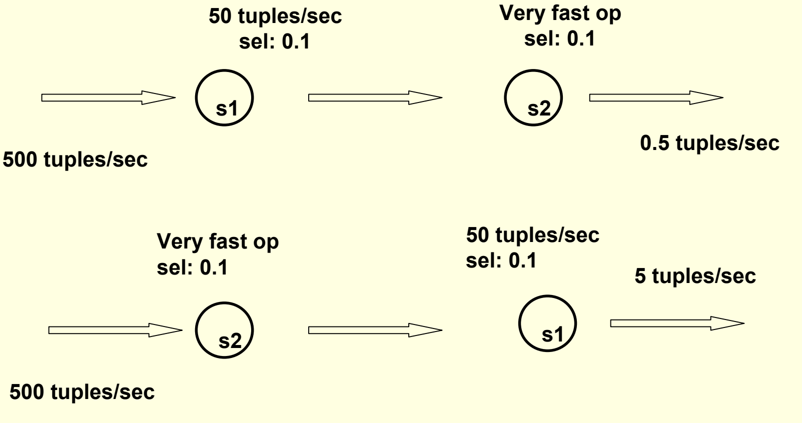
Optimization Objectives: Issues

- Traditionally table based cardinalities used in query optimization
- Problematic in a streaming environment
- Need for novel optimization objectives that are relevant when inputs consist of streaming information sources

Optimization Objectives

- Rate-based optimization [VN02]:
  - Take into account the rates of the streams in the query evaluation tree during optimization
  - Rates can be known and/or estimated
- Overall objective is to maximize the tuple output rate for a query
  - Instead of seeking the least cost plan, seek the plan with the highest tuple output rate.

Rate Based Optimization



Rate Based Optimization

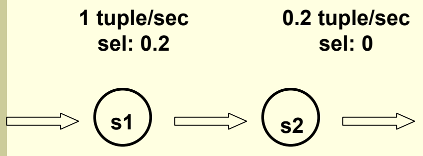
- Output rate of a plan: number of tuples produced per unit time
- Derive expressions for the rate of each operator
- Combine expressions to derive expression r(t) for the plan output rate as a function of time:
  - Optimize for a specific point in time in the execution
  - Optimize for the output production size

Optimization Objectives

- Optimize for resource (memory) consumption
- A query plan consists of interacting operators, with each tuple passing through a sequence of operators
- When streams are bursty, tuple backlog between operators may increase, affecting memory requirements
- Goal: scheduling policies that minimize resource consumption

Operator scheduling [BBDM03]

- When tuple arrival rate is uniform:
  - a simple FIFO scheduling policy suffices
  - let each tuple flow through the relevant operators



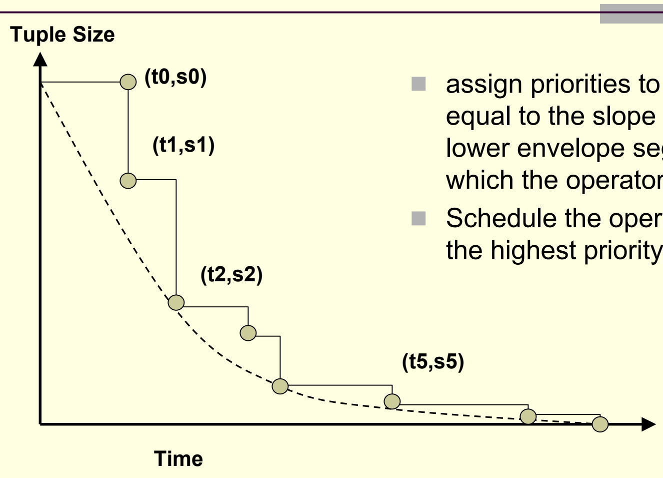
Time	Greedy	FIFO
0	1	1
1	1.2	1.2
2	1.4	2.0
3	1.6	2.2
4	1.8	3.0

Average arrival rate: 0.5 tuples/sec

FIFO: tuples processed in arrival order

Greedy: if tuple before s1 schedule it; otherwise process tuples before s2

Progress Chart: Chain Scheduling



- assign priorities to operators equal to the slope of the lower envelope segment to which the operator belongs
- Schedule the operator with the highest priority

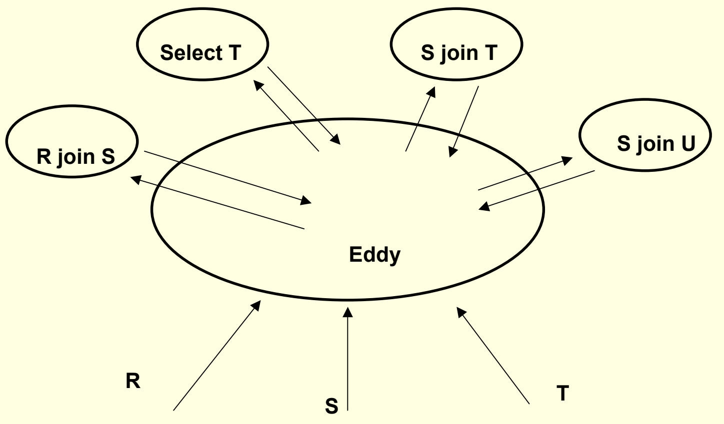
QoS Based Optimization [CCR+03]

- Query and operator scheduling based on performance and QoS requirements
- Two-level scheduling policy:
  - Operator batching: superbox selection, superbox traversal based on avg throughput, avg latency, minimizing memory
  - Tuple batching

Optimization Objectives

- Multi-way join techniques proposed:
  - start with a fixed plan
  - moderately adjust it as tuples arrive
- Eddies framework for adaptive query optimization:
  - Continuously adapt the evaluation order as tuples arrive

Eddies [AH00]



Optimization Objectives: Summary

- Novel notions of optimization
  - stream rate based
  - resource based
  - QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability

Load Shedding

- When input stream rate exceeds system capacity a stream manager can shed load (tuples)
- Load shedding affects queries and their answers
- Introducing load shedding in a data stream manager is a challenging problem
- Random and semantic load shedding

Load Shedding in Aurora [TCZ+03]

- QoS for each application as a function relating output to its utility
  - value based, loss tolerance based
- Techniques for introducing load shedding operators in a plan such that QoS is disrupted the least
  - Determining when, where and how much load to shed

Load Shedding in STREAM [BDM03]

- Formulate load shedding as an optimization problem for multiple sliding window aggregate queries
  - Minimize inaccuracy in answers subject to output rate matching or exceeding arrival rate
- Consider placement of load shedding operators in query plan
  - Each operator sheds load uniformly with probability pi

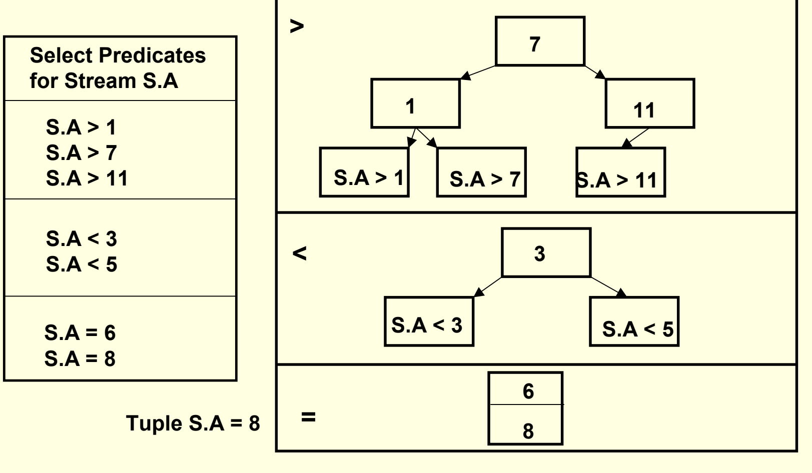
Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
  - Stream query language issues
  - Query operators
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: XML, open issues (45 min)

Multi-query Processing on Streams

- In traditional multi-query optimization:
  - sharing (of expressions, results etc) among queries can lead to improved performance
- Similar issues arise when processing queries on streams:
  - sharing between select/project expressions
  - sharing between sliding window join expressions

Grouped Filters [MSHR02]



Shared Window Joins [HFAE03]

- Consider the two queries:

select sum (A.length)
from Traffic1 A [window 1hour], Traffic2 B [window 1 hour]
where A.destIP = B.destIP

select count (distinct A.sourceIP)
from Traffic1 A [window 1 min], Traffic2 B [window 1 min]
where A.destIP = B.destIP

Shared Window Joins

- Great opportunity for optimization as windows are highly shared
- Strategies for scheduling the evaluation of shared joins:
  - Largest window only
  - Smallest window first
  - Process at any instant the tuple that is likely to benefit the largest number of joins (maximize throughput)

Shared Aggregations in Gigascope

- Multiple aggregate queries in a gigascope style query processing architecture.
- Issue with the allocation of buffer space at the lower level query processor of the architecture.
- Computing groups (phantoms) not explicitly requested in the query answer might be beneficial [ZKOS'05]

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
  - Stream query language issues
  - Query operators
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: XML, open issues (45 min)

Benchmark: Linear Road [ACG+04]

- Goal: Compare performance of DSMSs and DBMSs
- Linear Road Benchmark: Challenges
  - Semantically valid input: high-volume simulated data
  - Performance metrics: real-time query response, load
  - Many correct results: queries that depend on evolving state
  - No query language: queries specified in predicate calculus

Benchmark: Linear Road [ACG+04]

- Input data: Generated using MIT Traffic Simulator
  - Vehicles, accidents, congestion on multiple expressways
  - Position reports: (Time, VID, Spd, Xway, Lane, Dir, Seg, Pos)
  - Historical data: TollHistory, SegmentHistory
- Continuous query requests:
  - Toll notifications, toll assessments
  - Accident detection, accident notification
- Historical query requests:
  - Account balance: (Time, VID, QID)
  - Daily expenditure: (Time, VID, Xway, QID, Day)
  - Travel time: (Time, VID, Xway, QID, Sinit, Send, DOW, TOD)

Prototype systems

- Aurora (Brandeis, Brown, MIT) [CCC+02]
- Gigascope (AT&T) [CJSS03]
- Hancock (AT&T) [CFP+00]
- Nile (Purdue) [AEA+04]
- STREAM (Stanford) [MWA+03]
- Telegraph (Berkeley) [CCD+03]
- …

Aurora

- Geared towards monitoring applications (streams, triggers, imprecise data, real time requirements)
- Specified set of operators, connected in a data flow graph
- Optimization of the data flow graph
- Three query modes (continuous, ad-hoc, view)
- Aurora accepts QoS specifications and attempts to optimize QoS for the outputs produced
- Real time scheduling, introspection and load shedding

Gigascope

- Specialized stream database for network applications
- GSQL for declarative query specifications: pure stream query language (stream input/output)
- Uses ordering attributes in IP streams (timestamps and their properties) to turn blocking operators into non blocking ones
- GSQL processor is code generator.
- Query optimization uses a two level hierarchy

Hancock

- A C-based domain specific language which facilitates transactor signature extraction from transactional data streams
- Support for efficient and tunable representation of signature collections
- Support for custom scalable persistent data structures
- Elaborate statistics collection from streams

Nile

- Stream-in stream-out paradigm
- Summary Manager with the notion of promising tuples
- Sliding and predicate windows
- Negative tuples
- Shared execution
- Admission control and quality of service support
- Context-aware query processing and optimization
- Built-in online data mining
- Sensor network support
- Disk-based data streams

STREAM

- General purpose stream data manager
- CQL for declarative query specification
- Consider query plan generation
- Resource management:
  - operator scheduling
- Static and dynamic approximations

Telegraph

- Continuous query processing system
- Support for stream oriented operators
- Support for adaptivity in query processing
  - optimization
- Various aspects of optimized multi-query stream processing

Comparative Matrix

System	Data Stream Architecture	Data Model	Query Language	Query Answers	Query Plan
Aurora	low-level	RS-in RS-out	Operators	approximate	QoS-based, load shedding
Gigascope	two level (low, high)	S-in S-out	GSQL	exact	decomposition, avoid drops
Hancock	High-level	RS-in R-out	Procedural	exact, signatures	optimize for I/O, process blocks
STREAM	low-level	RS-in RS-out	CQL	approximate	optimize space, static analysis
Telegraph	high-level	RS-in RS-out	SQL-based	exact	adaptive plans, multi-query

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
- Part III: XML, open issues (45 min)
  - XML streams: motivation, query processing
  - XML streams: connection to tuple streams
  - Open Issues

XML Data Streams: Applications

- An **XML data stream** is a sequence of **tokens**
- Data and application integration
  - Distributed monitoring of computing systems
- Message-based web services
  - Purchase orders, retail transactions
- Personalized content delivery

XML Data Streams: Why XML?

- XML data model
  - Self-describing data representation
  - Natural for both structured and unstructured data
  - XML schema provides rich typing and structural constraints
- XML query languages
  - Declarative matching of structured data and text
  - Easy restructuring to meet needs of data consumers

Data Integration Application: Distributed Monitoring [MCC03]

- Business Challenge: Monitor the performance of a very large number (federation of clusters, grids) of computing devices
- Issues: Scalability, robustness, extensibility
- Solution: In Ganglia's hierarchical design
  - Monitoring daemons of single cluster report state in XML
  - Meta-daemons aggregate and export lower-level XML data
  - Permits queries based on path expressions, time windows
- Lesson: Essential to query distributed data streams

Web Services Application: Purchase Orders Workflow [FHK+03]

- Business Challenge: Automate enterprise business processes
- Issues: Need strong support for data transformations
- Solution: WLI uses XML and XQuery to
  - Transform purchase order data as it flows through system
  - Specify data-driven workflow logic (looping, branching)
- Lesson: Essential to transform and personalize data streams

WLI: Data Streams

- XML data: tree structure

```
<Purchase_Doc>
 <PR_Number val = “50”/>
 <Supp_Name>ABC</Supp_Name>
 <Address>
  <City>Florham Park</City>
  <State>New Jersey</State>
 </Address>
 <Line_Items>
  <Item>
   <Part_Number val= “1050”/>
   <Quantity val=“20”/>
  </Item>
```

- Data stream: ~ SAX events

```
[element Purchase_Doc anyType]
[element PR_Number anyType]
[attribute val anySimpleType]
[chardata 50]
[end-attribute]
[end-element]
[element Supp_Name anyType]
[text ABC]
[end-element]
…
```

WLI: Data Transformation Query

Purchase_Doc	Data ⇒ Transformation	Purchase_Req
PR_Number		PR_Number
Supp_Name		Line_Items
Address		Billing_Address
Line_Items		Address

```
<Purchase_Req> {
   for $d in document("doc.xml")/Purchase_Doc
   where $d//State = "New Jersey"
   return
      { $d/PR_Number } { $d/Line_Items }
      <Billing_Address> { $d/Address } </Billing_Address>
} </Purchase_Req>
```


XML Stream Processing: Key Ideas

- Obtain bindings of **for** clause path expression variables
  - Ordered sequence, no duplicates
- Filter bindings using **where** clause path expression predicates
  - Existential check suffices
- Compute bindings of **return** clause path expressions
  - Ordered (possibly null) sequence

- Goal: Efficient matching/binding of XML path expressions
  - Very large number of path expressions

Result Generation: Alternatives

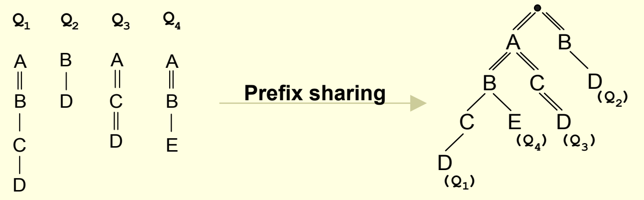
Path Matching	Result Generation
for clause paths	Navigation for **where** clause paths Navigation for **return** clause paths
for-where clause paths	Semijoins for **where** clause paths Navigation for **return** clause paths
for-where-return clause paths	Semijoins for **where** clause paths Left outerjoins for **return** clause paths

Shared Path Matching: Overview

- Automata-based techniques:
  - XFilter [AF00]: finite state machine per path expression
  - XTrie [CFGR02]: shares common sub-paths of PC paths
  - **YFilter** [DF03]: single NFA for all path expressions
  - [GMOS03]: single DFA, limitations on flexibility
  - XPush [GS03]: pushdown automaton for tree patterns
- Index-based techniques:
  - MatchMaker [LP02]: shared tree patterns
  - **IndexFilter** [BGKS03]: shared path expressions, comparison

Prefix Sharing for Path Expressions

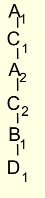
- Used by automata- and index-based techniques
- Eliminates redundant processing
- Tradeoff between “compression” and “amount of bookkeeping”



Set of Path Expressions — Path Prefix Tree

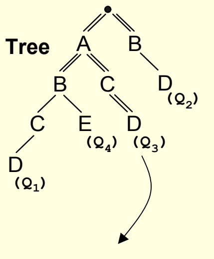
Solution Representation

[element A anyType]
[element C anyType]
[element A anyType]
[element C anyType]
[element B anyType]
[element D anyType]
[end-element]
[end-element]
[end-element]
[end-element]
[end-element]
[end-element]



XML Data

Path Prefix Tree

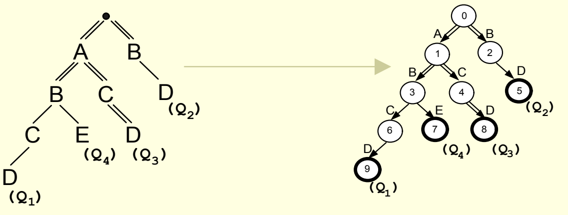


A	C	D
A_1	C_1	D_1
A_1	C_2	D_1
A_2	C_2	D_1

Matches

YFilter: Automata-based Technique

- Augment the path prefix tree into an NFA



- Implemented as a tree of hash tables for efficiency
- Set of active states maintained in a runtime stack
  - Adds sets of states to stack when encountering open-tags
  - Backtracks stack when encountering close-tags

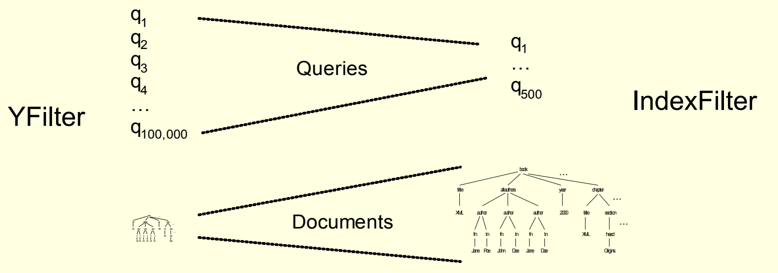
IndexFilter: Index-based Technique

- Build and exploit (start, end, level) indexes on XML tags

- Each node **q** in path prefix tree contains:
  - T_q: Indexed stream of tags sorted by “start”
  - S_q: Stack to track all matches
  - P_q: Dynamic access to q’s children with minimal start value

- Tradeoff index building cost with unnecessary matching in NFA
  - Use of indexes focuses matching to relevant tags

YFilter versus IndexFilter: Summary



- Other relevant factors:
  - Document/query structure, selectivity of path expressions

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
- Part III: XML, open issues (45 min)
  - XML streams: motivation, query processing
  - XML streams: connection to tuple streams
  - Open Issues

Connections

XML Streams	Tuple Streams
One heterogeneous stream of XML tokens	Multiple homogeneous sub-streams of tuples
(Shared) path matching	(Shared) multi-way joins
Explicit [end-element] and [end-attribute] tokens	Explicit punctuations to identify “end of processing”

Sub-streams of an XML Stream

[element Address]		[element A, 30, 2]				
[element City]			[element C, 31, 3]			
[text FP]					[text FP, 32, 4]	
[end-element]						[end-elem, 33, 31]
[element State]				[element S, 34, 3]		
[text NJ]	⇒				[text NJ, 35, 4]	
[end-element]						[end-elem, 36, 34]
[end-element]						[end-elem, 37, 30]
…						…
[element Address]		[element A, 65, 2]				
[element City]			[element C, 66, 3]			
[text MT]					[text MT, 67, 4]	
[end-element]						[end-elem, 68, 66]
[element State]				[element S, 69, 3]		
[text NJ]					[text NJ, 70, 4]	
[end-element]						[end-elem, 71, 69]
[end-element]						[end-elem, 72, 65]

Path Matching as Multi-way Joins

- Path: //A/S[. = "NJ"]
- SQL query:

```
select A.id, S.id
from A, S, text T, end E E1
where S.id > A.id AND
   S.level = A.level+1 AND
   T.id > S.id AND
   T.level = S.level+1 AND
   T.value = "NJ" AND
   E.id > T.id AND
   E.back = S.id AND
   E1.id > E.id AND
   E1.back = A.id
```

[element A, 30, 2]			
		[text FP, 32, 4]	[end-elem, 33, 31]
	[element S, 34, 3]	[text NJ, 35, 4]	[end-elem, 36, 34]
			[end-elem, 37, 30]
[element A, 65, 2]			…
		[text MT, 67, 4]	[end-elem, 68, 66]
	[element S, 69, 3]	[text NJ, 70, 4]	
			[end-elem, 71, 69]
			[end-elem, 72, 65]

End-element as Punctuation

- Limits query scope
- Nested structure

```
select A.id, S.id
from A, S, text T, end E E1
where S.id > A.id AND
   S.level = A.level+1 AND
   T.id > S.id AND
   T.level = S.level+1 AND
   T.value = "NJ" AND
   E.id > T.id AND
   E.back = S.id AND
   E1.id > E.id AND
   E1.back = A.id
```

[element A, 30, 2]			
		[text FP, 32, 4]	[end-elem, 33, 31]
	[element S, 34, 3]	[text NJ, 35, 4]	[end-elem, 36, 34]
			[end-elem, 37, 30]
[element A, 65, 2]			…
		[text MT, 67, 4]	[end-elem, 68, 66]
	[element S, 69, 3]	[text NJ, 70, 4]	
			[end-elem, 71, 69]
			[end-elem, 72, 65]

Stream Map

- Part I: Motivation (45 min)
- Part II: Query processing (90 min)
- Part III: XML, open issues (45 min)
  - XML streams: motivation, query processing
  - XML streams: connection to tuple streams
  - Open Issues

Open Issues: Multi-way Joins

- Recent work on multi-way joins with equality predicates
- Issue: How do we use constraints and punctuations effectively?
  - Self-joins: looking for patterns in a single stream
  - Natural constraints other than "nesting" constraints
- Current status:
  - XML path matching is a special case

Open Issues: Approximate Aggregates

- Large body of work on approximate aggregates over streams
- Issue: How can this work be used by data stream systems?
  - Engineering summary structures (sketches, samples) for low-level data stream processing
- Current status: Quantile computation is part of Gigascope, and engineered to reduce drops

Open Issues: Query Decomposition

- End-to-end three-level architecture:
  - Low-level and high-level data streams, DBMS
- Issue: How do we decompose a declarative (SQL) query?
  - Need to take resource limitations at each level into account
  - Which sub-queries are evaluated by which level?
- Current status: Gigascope does some automatic decomposition, and provides hooks for manual decomposition

Open Issues: Distributed Evaluation

- Low-level data stream processing may be highly distributed
- Issue: How do we correlate distributed data streams?
  - May not be feasible to bring all relevant data to a single site
  - Can one use techniques from distributed DBMSs?
- Current status: Some preliminary work by Aurora and STREAM people [BO03, CBB+03, OJW03]

Open Issues: Query Optimization

- Data stream properties (arrival rate, sortedness) may vary a lot
- Issue: How do we evaluate queries efficiently?
  - Adaptive strategies like Eddies have high overheads, and may not be directly feasible for data stream systems
  - Can one borrow ideas from queuing theory?
- Current status: Preliminary cost models developed by various data stream projects

Open Issues: I/O and Streaming

- High-level data stream processing can populate DBMS
- Issue: How do we process streams to minimize DBMS I/O?
  - Need to process streams in blocks, using multiple passes
  - How can multiple streams be correlated for this purpose?
- Current status: Hancock pays attention to I/O issues when computing signatures, other stream systems do not focus on I/O

Conclusions

- Data stream query processing has real applications
  - Need for sophisticated near-real time queries
  - Massive data volumes of transactions and measurements
- Wealth of challenging technical problems
  - Resource limitations exist, especially at low-level
  - Important to think of the end-to-end architecture

References

- [AF00] M. Altinel, M. J. Franklin: Efficient Filtering of XML Documents for Selective Dissemination of Information. VLDB 2000: 53-64
- [AFTU96] L. Amsaleg, M. J. Franklin, A. Tomasic, T. Urhan: Scrambling Query Plans to Cope With Unexpected Delays. PDIS 1996: 208-219
- [ABB+02] A. Arasu, B. Babcock, S. Babu, J. McAlister, J. Widom: Characterizing Memory Requirements for Queries over Continuous Data Streams. PODS 2002: 221-232
- [ABB+03] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, J. Widom: STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin 26(1): 19-26 (2003)
- [ABW03] A. Arasu, S. Babu, J. Widom: An Abstract Semantics and Concrete Language for Continuous Queries Over Data Streams. DBPL 2003
- [ACG+04] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, R. Tibbetts: Linear Road: A Stream Data Management Benchmark. VLDB 2004
- [AEA+04] W. Aref, A. Elmargamid, M. Ali, M. Caltin et. Al.: Nile: A Query Processing Engine for Data Streams, ICDE 2004.
- [AH00] R. Avnur, J. M. Hellerstein: Eddies: Continuously Adaptive Query Processing. SIGMOD Conference 2000: 261-272
- [BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom: Models and Issues in Data Stream Systems. PODS 2002: 1-16
- [BBDM03] B. Babcock, S. Babu, M. Datar, R. Motwani: Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. SIGMOD Conference 2003: 253-264
- [BDM03] B. Babcock, M. Datar, R. Motwani: Load Shedding Techniques for Data Stream Systems. MPDS Workshop 2003
- [BO03] B. Babcock, C. Olston: Distributed Top-K Monitoring. SIGMOD Conference 2003: 28-39

References

- [BW01] S. Babu, J. Widom: Continuous Queries over Data Streams. SIGMOD Record 30(3): 109-120 (2001)
- [BMMNW04] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, J. Widom. Adaptive Ordering of Pipelined Stream Filters, SIGMOD 2004: 407-418.
- [BR87] I. Balbin, K. Ramamohanarao: A Generalization of the Differential Approach to Recursive Query Evaluation. JLP 4(3): 259-262 (1987)
- [BDF+97] D. Barbara, W. DuMouchel, Christos Faloutsos, Peter J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, K. C. Sevcik: The New Jersey Data Reduction Report. Data Engineering Bulletin 20(4): 3-45 (1997)
- [BCG+03] C. Barton, P. Charles, D. Goyal, M. Raghavachari, M. Fontoura, V. Josifovski: Streaming XPath Processing with Forward and Backward Axes. ICDE 2003
- [BFRS99] D. Bonachea, K. Fisher, A. Rogers, F. Smith: Hancock: a language for processing very large-scale data. DSL 1999: 163-176
- [BGKS03] N. Bruno, L. Gravano, N. Koudas, D. Srivastava: Navigation- vs. Index-Based XML Multi-Query Processing. ICDE 2003
- [CCC+02] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. B. Zdonik: Monitoring Streams - A New Class of Data Management Applications. VLDB 2002: 215-226
- [CCR+03] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, M. Stonebraker: Operator Scheduling in a Data Stream Manager. VLDB 2003
- [CFGR02] C. Y. Chan, P. Felber, M. N. Garofalakis, R. Rastogi: Efficient filtering of XML documents with XPath expressions. VLDB Journal 11(4): 354-379 (2002)
- [CF02] S. Chandrasekaran, M. J. Franklin: Streaming Queries over Streaming Data. VLDB 2002: 203-214

References

- [CCD+03] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, M. A. Shah: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR 2003
- [CR96] D. Chatziantoniou, K. A. Ross: Querying Multiple Features of Groups in Relational Databases. VLDB 1996: 295-306
- [CDTW00] J. Chen, D. J. DeWitt, F. Tian, Y. Wang: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. SIGMOD Conference 2000: 379-390
- [CBB+03] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. B. Zdonik: Scalable Distributed Stream Processing. CIDR 2003
- [CFP+00] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, F. Smith: Hancock: a language for extracting signatures from data streams. KDD 2000: 9-17
- [CJSS03] C. D. Cranor, T. Johnson, O. Spatscheck, V. Shkapenyuk: Gigascope: A Stream Database for Network Applications. SIGMOD Conference 2003: 647-651
- [CJSS03a] C. D. Cranor, T. Johnson, O. Spatscheck, V. Shkapenyuk: The Gigascope Stream Database. IEEE Data Engineering Bulletin 26(1): 27-32 (2003)
- [DF03] Y. Diao, M. J. Franklin: High-Performance XML Filtering: An Overview of YFilter. IEEE Data Engineering Bulletin 26(1): 41-48 (2003)
- [DF03a] Yanlei Diao, Michael J. Franklin: Query Processing for High-Volume XML Message Brokering. VLDB 2003
- [DMRH'04] L. Ding, N. Mehta, E. Rundersteiner, G. Heineman: Joining Punctuated Streams EDBT 2004.

References

- [FLBC02] L. Fegaras, D. Levine, S. Bose, V. Chaluvadi: Query Processing of Streamed XML Data. CIKM 2002
- [FHK+03] D. Florescu, C. Hillary, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, A. Sundararajan, G. Agrawal: A Complete and High-performance XQuery Engine for Streaming Data. VLDB 2003
- [GGR02] Minos N. Garofalakis, Johannes Gehrke, Rajeev Rastogi: Querying and Mining Data Streams: You Only Get One Look: A Tutorial. SIGMOD Conference 2002: 635
- [GO03] L. Golab, M. T. Ozsu: Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. VLDB 2003
- [GMOS03] T. J. Green, G. Miklau, M. Onizuka, D. Suciu: Processing XML Streams with Deterministic Automata. ICDT 2003
- [GS03] A. Gupta, D. Suciu: Stream Processing of XPath Queries with Predicates. SIGMOD Conference 2003
- [HFAE03] M. A. Hammad, M. J. Franklin, W. G. Aref, A. K. Elmagarmid: Scheduling for shared window joins over data streams. VLDB 2003
- [ILW00] Z. Ives, A. Y. Levy, D. Weld: Efficient evaluation of regular path expressions on streaming data. University of Washington Tech Report, 2000
- [JMS95] H. V. Jagadish, I. S. Mumick, A. Silberschatz: View Maintenance Issues for the Chronicle Data Model. PODS 1995: 113-124

References

- [KNV03] J. Kang, J. F. Naughton, S. Viglas: Evaluating window joins over unbounded streams. ICDE 2003: 37-48
- [LP02] L. V. S. Lakshmanan, S. Parthasarathy: On Efficient Matching of Streaming XML Documents and Queries. EDBT 2002: 142-160
- [LCHT02] M-L. Lee, B. C. Chua, W. Hsu, K-L. Tan: Efficient Evaluation of Multiple Queries on Streaming XML Data. CIKM 2002
- [LPT99] L. Liu, C. Pu, W. Tang: Continual Queries for Internet Scale Event-Driven Information Delivery. TKDE 11(4): 610-628 (1999)
- [LWZ04] Y. N. Law, H. Wang, C. Zaniolo: Query Languages and Data Models for Database Sequences and Data Streams. VLDB 2004
- [MF02] S. Madden, M. J. Franklin: Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. ICDE 2002: 555-566
- [MSHR02] S. Madden, M. A. Shah, J. M. Hellerstein, V. Raman: Continuously adaptive continuous queries over streams. SIGMOD Conference 2002: 49-60
- [MFHH03] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong: The Design of an Acquisitional Query Processor For Sensor Networks. SIGMOD Conference 2003: 491-502
- [MCC03] M. L. Massie, B. N. Chun, D. E. Culler: The Ganglia Distributed Monitoring System: Design, Implementation and Experience. Draft, 2003. See http://ganglia.sourceforge.net/
- [MWA+03] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, R. Varma: Query Processing, Approximation, and Resource Management in a Data Stream Management System. CIDR 2003
- [MP80] J. I. Munro, M. Paterson: Selection and Sorting with Limited Storage. TCS 12: 315-323 (1980)

References

- [M03] S. Muthukrishnan: Data streams: algorithms and applications. SODA 2003: 413-413 http://athos.rutgers.edu/~muthu/stream-1-1.ps
- [MS03] S. Muthukrishnan, D. Srivastava: Workshop on Management and Processing of Data Streams (2003). http://www.research.att.com/conf/mpds2003/
- [OJW03] C. Olston, J. Jiang, J. Widom: Adaptive Filters for Continuous Queries over Distributed Data Streams. SIGMOD Conference 2003: 563-574
- [PC03] F. Peng, S. S. Chawathe: XPath Queries on Streaming Data. SIGMOD Conference 2003: 431-442
- [PFJ+01] J. Pereira, F. Fabret, H-A. Jacobsen, F. Llirbat, D. Shasha: WebFilter: A High-throughput XML-based Publish and Subscribe System. VLDB 2001: 723-724
- [RNC03] G. Russell, M. Neumuller, R. Connor: TypEx: A Type Based Approach to XML Stream Querying. WebDB 2003
- [SS96] S. Sarawagi, M. Stonebraker: Reordering Query Execution in Tertiary Memory Databases. VLDB 1996: 156-167
- [SV02] L. Segoufin, V. Vianu: Validating Streaming XML Documents. PODS 2002: 53-64
- [SLR94] P. Seshadri, M. Livny, R. Ramakrishnan: Sequence Query Processing. SIGMOD Conference 1994: 430-441
- [SLR95] P. Seshadri, M. Livny, R. Ramakrishnan: SEQ: A Model for Sequence Databases. ICDE 1995: 232-239
- [S96] M. Sullivan: Tribeca: A Stream Database Manager for Network Traffic Analysis. VLDB 1996: 594

References

- [TCG+93] A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, R. T. Snodgrass: Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings 1993
- [TCZ+03] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, M. Stonebraker: Load Shedding in a Data Stream Manager. VLDB 2003
- [TGNO92] D. B. Terry, D. Goldberg, D. Nichols, B. M. Oki: Continuous Queries over Append-Only Databases. SIGMOD Conference 1992: 321-330
- [TMSF03] P. A. Tucker, D. Maier, T. Sheard, L. Fegaras: Exploiting Punctuation Semantics in Continuous Data Streams. TKDE 15(3): 555-568 (2003)
- [TMS03] P. A. Tucker, D. Maier, T. Sheard: Applying Punctuation Schemes to Queries Over Continuous Data Streams. IEEE Data Engineering Bulletin 26(1): 33-40
- [UF00] T. Urhan, M. J. Franklin: XJoin: A Reactively-Scheduled Pipelined Join Operator. IEEE Data Engineering Bulletin 23(2): 27-33 (2000)
- [VN02] S. Viglas, J. F. Naughton: Rate-based query optimization for streaming information sources. SIGMOD Conference 2002: 37-48
- [VNB03] S. Viglas, J. F. Naughton, J. Burger: Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. VLDB 2003
- [WA91] A. N. Wilschut, P. M. G. Apers: Dataflow Query Execution in a Parallel Main-Memory Environment. PDIS 1991: 68-77
- [ZGTS02] D. Zhang, D. Gunopulos, V. J. Tsotras, B. Seeger: Temporal Aggregation over Data Streams Using Multiple Granularities. EDBT 2002: 646-663