Stream Data Management

Divesh Srivastava
AT&T Labs-Research
http://www.research.att.com/~divesh/

Stream Map

- Part I: Motivation
  - Data streams: what, why now, applications
  - Data streams: architecture and issues
- Part II: Query processing
- Part III: Gigascope DSMS

Data Streams: What and Where?

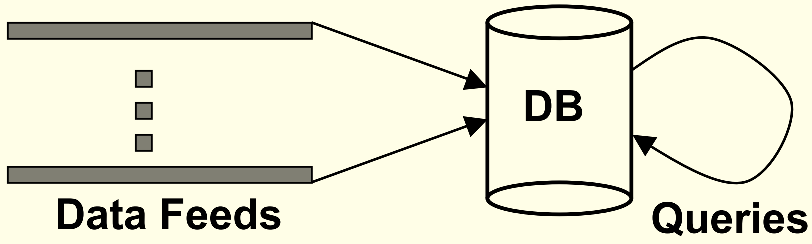
- A **data stream** is a (potentially unbounded) sequence of tuples

- **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
  - Finance: stock prices, bids and asks

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

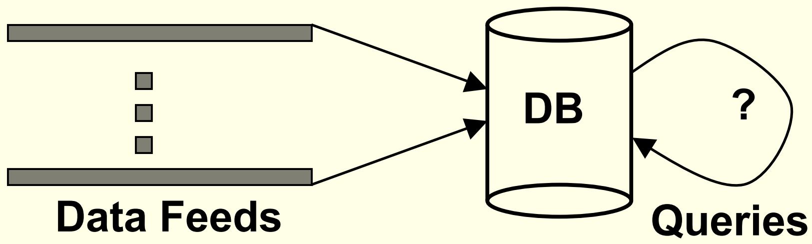


- Two recent developments: application- and technology-driven
  - Need for sophisticated near-real time queries/analyses
  - Massive data volumes of transactions and measurements

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

- Issues:
  - Handwritten, optimized C code difficult to maintain
  - Signature computation is I/O intensive

- Solution: Using Hancock domain-specific language
  - Abstract logical/physical streams and signatures
  - Express I/O and CPU efficient signature programs cleanly

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

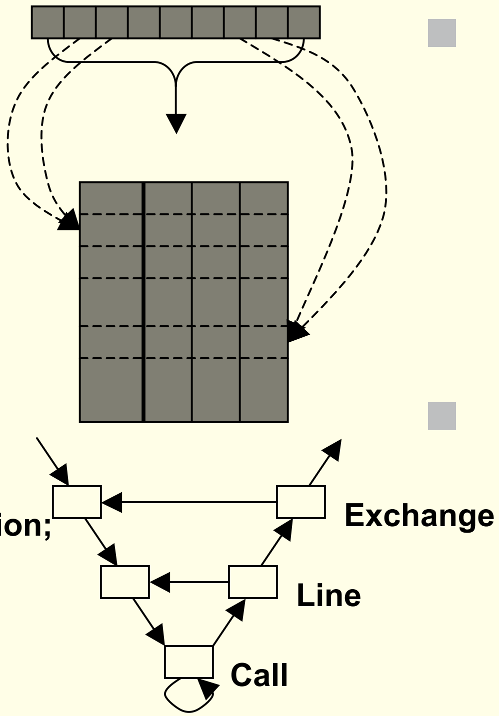
Hancock: Signature Programs

```
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;
}}
```

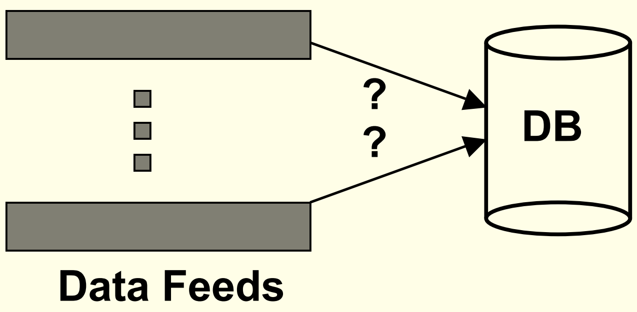


- Hancock program paradigm:
  - Stream-in, relation-out
  - Block processing of data
  - Multiple passes on block
- Hancock programs support:
  - Iterating on sorted data
  - Filtering
  - Event clause hierarchy
- User-defined aggregation

Data Streams: Massive Volumes

- Now able to deploy transactional data observation points
  - AT&T long-distance: ~300M calls/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

- Issues
  - Use of P2P port numbers in Netflow data is not adequate
  - P2P traffic may be “hidden” in, e.g., HTTP traffic

- Solution: Using Gigascope data stream management system
  - Search for P2P related keywords within TCP datagrams
  - Classified 3 times more traffic as P2P than Netflow

IP Network Application: Web Client Performance Monitoring

- Business Challenge: AT&T IP customer wanted to monitor latency observed by clients to find performance problems

- Issues
  - Use of few “active clients” is not very representative
  - Massive volumes of data (Gbit/sec links, multiple links)

- Solution: Using Gigascope data stream management system
  - Track timestamps of TCP SYN and ACK packets
  - Report latency as RTT, i.e., difference of timestamps

IP Network Application: Security

- Business Challenge: Alert IP customers about DDoS attacks and worms by monitoring and analyzing network data streams

- Issues
  - Massive volumes of data (Gbit/sec links, multiple links)
  - Real-time alerting (reaction time in minutes, not days)

- Solution: Using Gigascope data stream management system
  - Monitor IP traffic data streams across customer networks
  - Analyze headers + contents, identify new attack signatures

IP Network Packet Data

```
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;
}
```

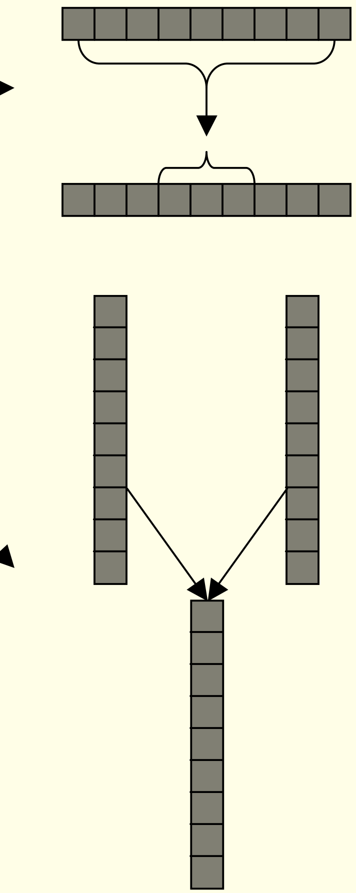
- Heterogeneous records
  - Layer 2: ETH/HDLC
  - Layer 3: IP/IPv4
  - Layer 4: UDP/TCP/ICMP
  - Layers 5-7: application level, e.g., HTTP, SMTP
- Analysis complicated by
  - Missing packets
  - Repeated packets
  - Out of order packets

Gigascope

- Gigascope is a fast, flexible data stream management system
  - High performance at speeds up to OC768 (2 x 40 Gbits/sec)
  - GSQL queries support SQL-like functionality
- Monitoring platform of choice for AT&T IP network
- Developed at AT&T Labs-Research
  - Collaboration between database and networking research

Gigascope: GSQL Queries

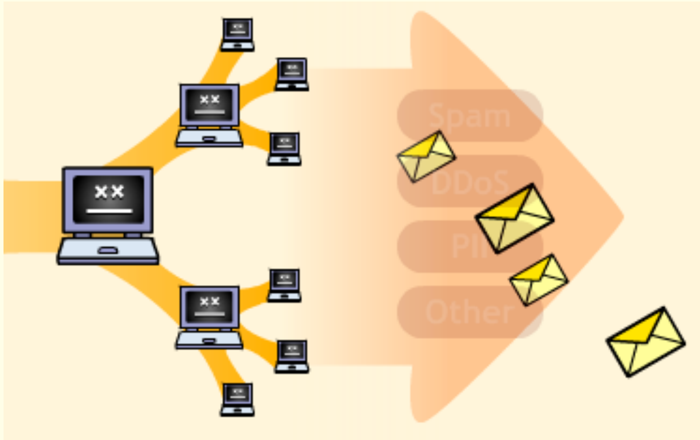
- GSQL queries support:
  - Filtering, aggregation
  - Merges and joins
- Arbitrary code support
  - UDFs (e.g., LPM)
  - UDAFs
- GSQL query paradigm:
  - Streams-in, stream-out
  - Permits composability



Example: Email Bombing

- Attack characteristic: excessively many email messages
- Attack detection: monitor SMTP traffic, compare with trends
- GSQL query

```
define { query_name smtp_perhost; }
select tb, destIP, count(*), sum(len)
from TCP
where protocol = 6 and destPort = 25
group by time/60 as tb, destIP
```



Example: TCP SYN Flood

- Attack characteristic: exploits 3-way TCP handshake
- Attack detection: correlate SYN, ACK packets in TCP stream
- GSQL query

```
define { query_name toomany_syn; }
select A.tb, (A.cnt – M.cnt)
outer_join from all_syn_count A,
   matched_syn_count M
where A.tb = M.tb

define { query_name all_syn_count; }
select S.tb, count(*) as cnt
from tcp_syn S
group by S.tb
```

```
define { query_name matched_syn_count; }
select S.tb, count(*) as cnt
from tcp_syn S, tcp_ack A
where S.sourceIP = A.destIP and
   S.destIP = A.sourceIP and
   S.sourcePort = A.destPort and
   S.destPort = A.sourcePort and
   S.tb = A.tb and
   S.timestamp <= A.timestamp and
   (S.sequence_number+1) = A.ack_number
group by S.tb
```

Example: Port Scans

- Attack characteristic: probing for vulnerability
- Attack detection: track number of distinct targets probed
- GSQL query

```
define { query_name
   countdest_persource; }
select tb, sourceIP, count_distinct(
   PACK(destIP,destPort) ) as cnt
from TCP
group by time/60 as tb, sourceIP
```

```
define { query_name countdest; }
select tb, count_distinct(
   PACK(destIP,destPort) ) as cnt
from TCP
group by time/60 as tb
```

- Illustrates use of UDAFs, approximate algorithms

Example: Worms

- Attack characteristic: self-propagating malicious code
- Attack detection: payload analysis using inverse distributions
- GSQL query

```
define { query_name inverse_distrib; }
select B.tb, B.cnt, count(*) as invcnt
from base_distrib B
group by B.tb, B.cnt
```

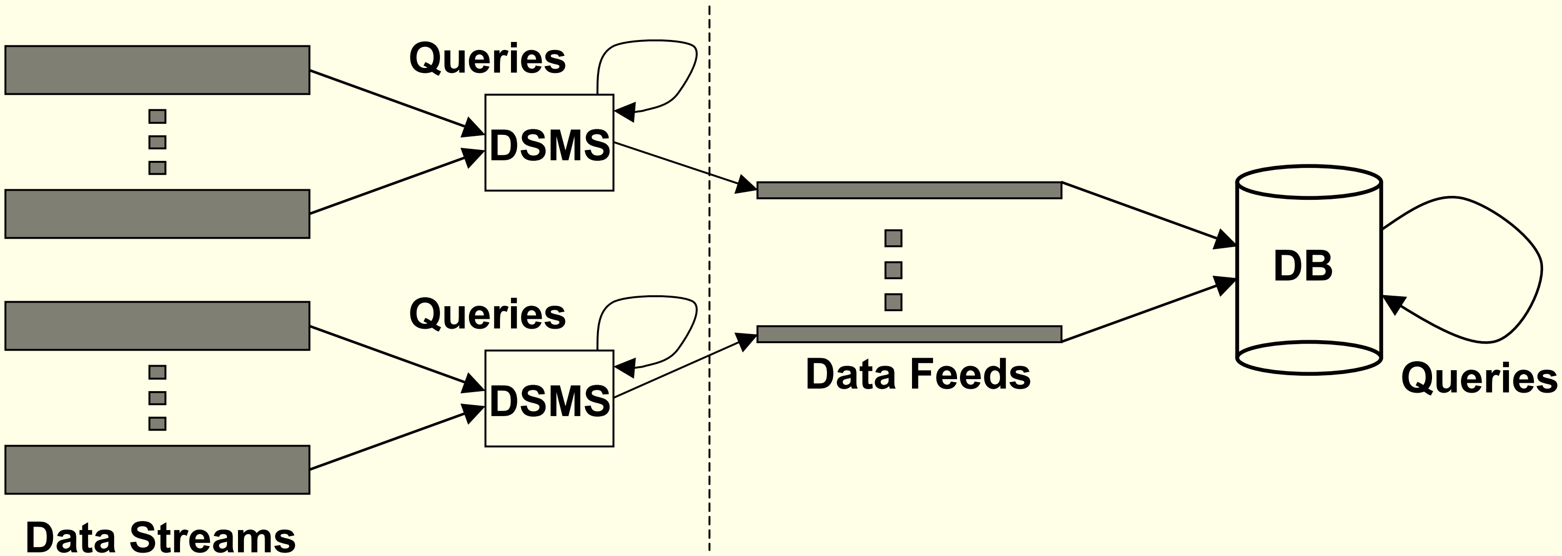
```
define { query_name base_distrib; }
select C.tb, C.Sid, count(*) as cnt
from tcp_content C
group by C.tb, C.Sid
```

Stream Map

- Part I: Motivation
  - Data streams: what, why now, applications
  - Data streams: architecture and issues
- Part II: Query processing
- Part III: Gigascope DSMS

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	Database Systems
Resource (memory, per-tuple computation) limited	Resource (memory, disk, per-tuple computation) rich
Reasonably complex, near real time, query processing	Extremely sophisticated query processing, analyses
Useful to identify what data to populate in database	Useful to audit query results of data stream system

DBMS versus DSMS: Issues

Database Systems	Data Stream Systems
Model: persistent relations	Model: transient relations
Relation: tuple set/bag	Relation: tuple sequence
Data Update: modifications	Data Update: appends
Query: transient	Query: persistent
Query Answer: exact	Query Answer: approximate
Query Evaluation: arbitrary	Query Evaluation: one pass
Query Plan: fixed	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

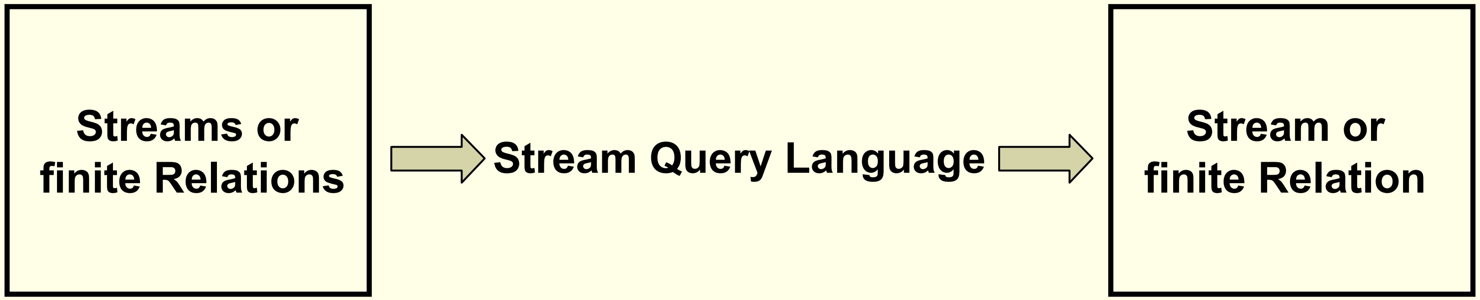
A lot of challenging problems ...

Stream Map

- Part I: Motivation
- Part II: Query processing
  - Stream query language issues (compositionality, windows)
  - Query operators
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: Gigascope DSMS

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

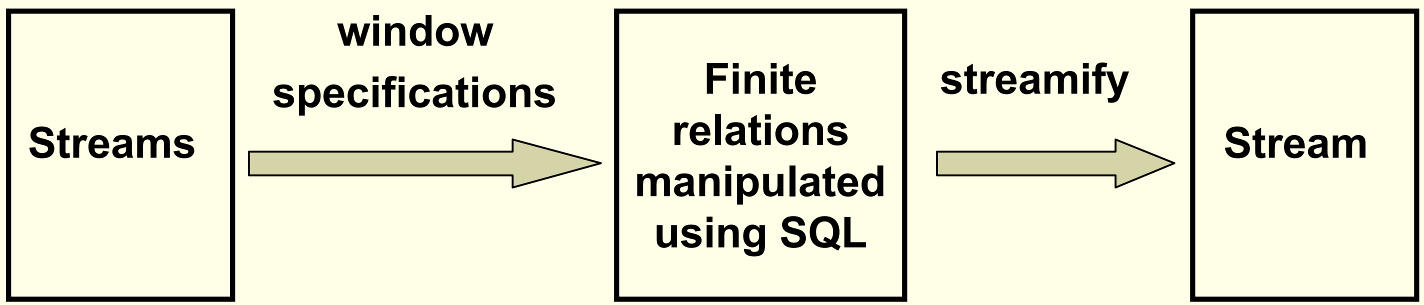


- UDA-SQL [LWZ04]: Monotonic sequence based queries

Windows

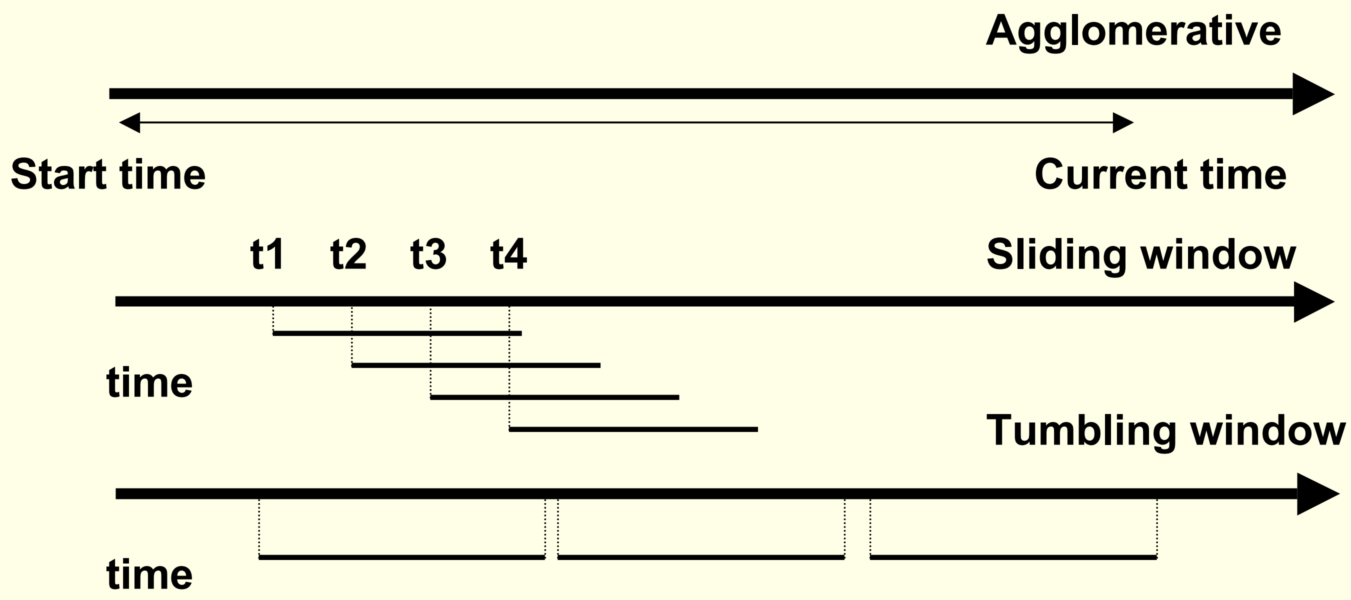
- Mechanism for extracting a finite relation from an infinite stream

- 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)



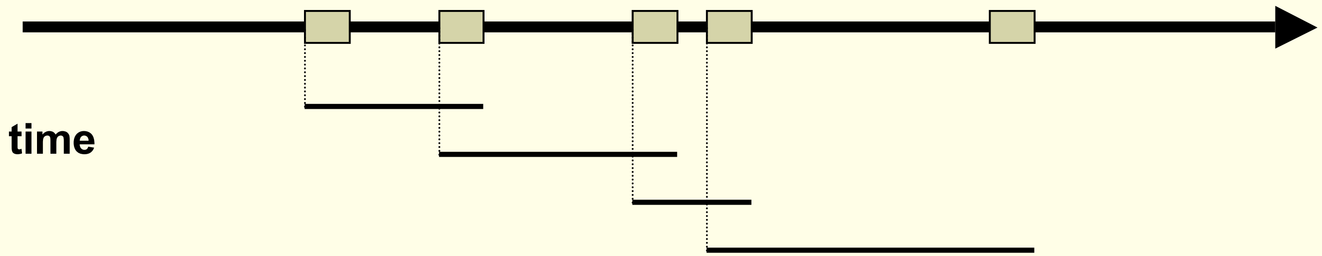
Ordering Attribute Based Windows

- Assumes existence of an ordering attribute (e.g., time)
- 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
- Part II: Query processing
  - Stream query language issues
  - Query operators (selections/projections, joins, aggregations)
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: Gigascope DSMS

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 TCP
where length > 512

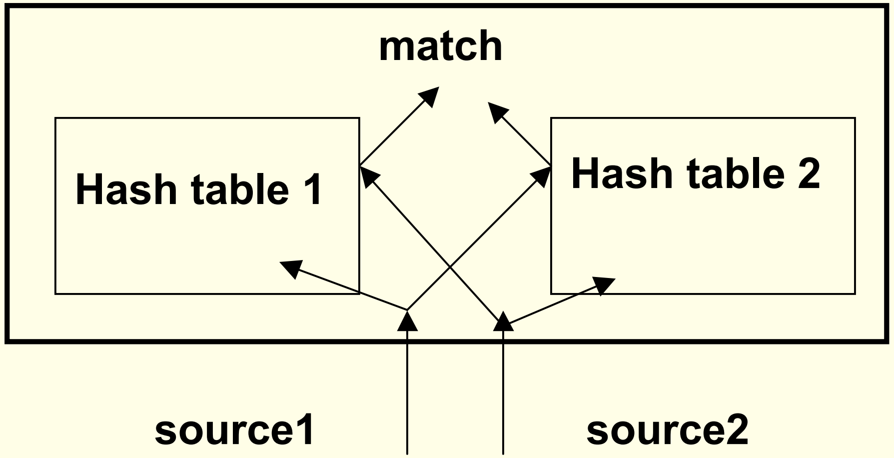
Join Operators

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

```
Select A.sourceIP, B.sourceIP
from TCP A [window T1], TCP B [window T2]
where A.destIP = B.destIP
```

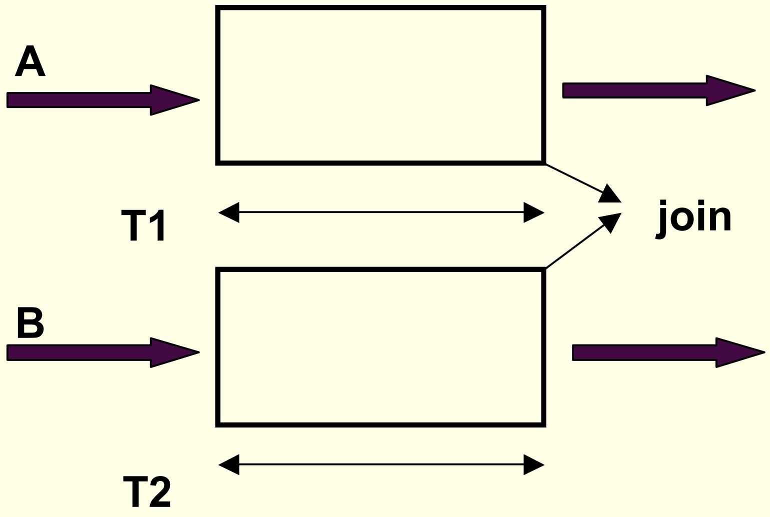
Join Operators: Background

- 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

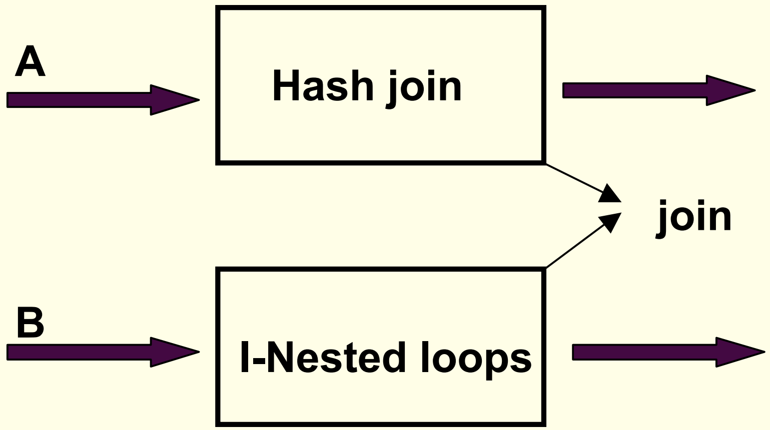
Binary Joins [KNV03]



New A tuple:

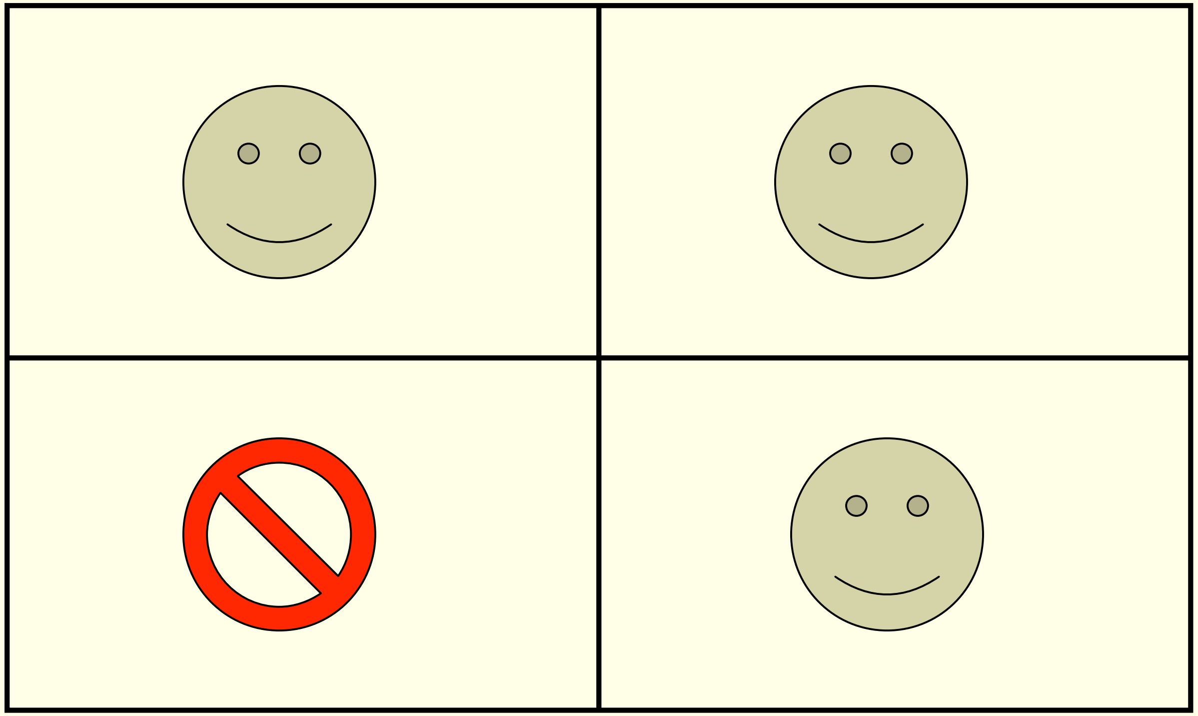
- Scan B's window for joining tuples and output result
- Insert tuple into A's window
- **Invalidate** all expired tuples in A's window

Binary Joins: Asymmetry



- Asymmetric join processing useful if arrival rates differ
- Goal: maximize tuple output
  - Limited computation, but sufficient memory
  - Limited memory, but sufficient computation

Strategies and Expirations



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 TCP [window T]
where length > 512
group by length
```

≡

```
select distinct length
from TCP [window T]
where length > 512
```

- Aggregate query execution in bounded memory:

```
select length, count(*)
from TCP [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 TCP group by time/60 as tb, f(destIP,’peerid.tbl’) as peerid
  - time/60 is a minute-long tumbling window (epoch)

- 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

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
- Use summary structures: samples, histograms, sketches

Quantiles

- What: quantiles are order statistics
  - Minimum, maximum, median
  - Φ-quantile: item with rank ΦN in data set of size N
- Why: useful to summarize data distributions
  - Example: 0.1, 0.2, …, 0.9-quantiles of GRE scores
  - Median (0.5-quantile) more robust to outliers than average

Quantile Computation

- Exact computation of Φ-quantile
  - Sort data set, pick out item in position ΦN
  - On a data stream (one pass), need $\Omega(N)$ space [MP80]

- ε-approximate computation in sub-linear space
  - Φ-quantile: item with rank between $(\Phi-\varepsilon)N$ and $(\Phi+\varepsilon)N$
  - [MRL98]: N known a priori, space $O(1/\varepsilon \log^2(\varepsilon N))$
  - [GK01]: N not known a priori, space $O(1/\varepsilon \log(\varepsilon N))$

Biased Quantiles: Motivation

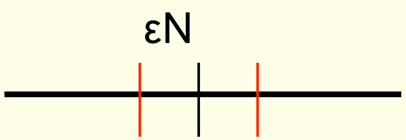
- IP network traffic has a lot of skew
  - Long tails of great interest
  - Example: 0.9, 0.95, 0.99-quantiles of TCP round trip times

- Issue: uniform error guarantees
  - $\varepsilon = 0.05$: okay for median, but not 0.99-quantile
  - $\varepsilon = 0.001$: okay for both, but needs too much space

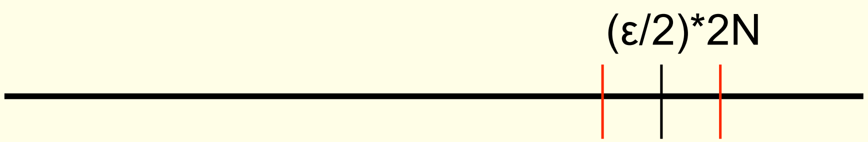
- Goal: support relative error guarantees in small space
  - $1-\Phi, \ldots, 1-\Phi^k$ quantiles in ranks $(1-(1\pm\varepsilon)\Phi)N, \ldots, (1-(1\pm\varepsilon)\Phi^k)N$

Biased Quantiles: Intuition

- Median at time step N



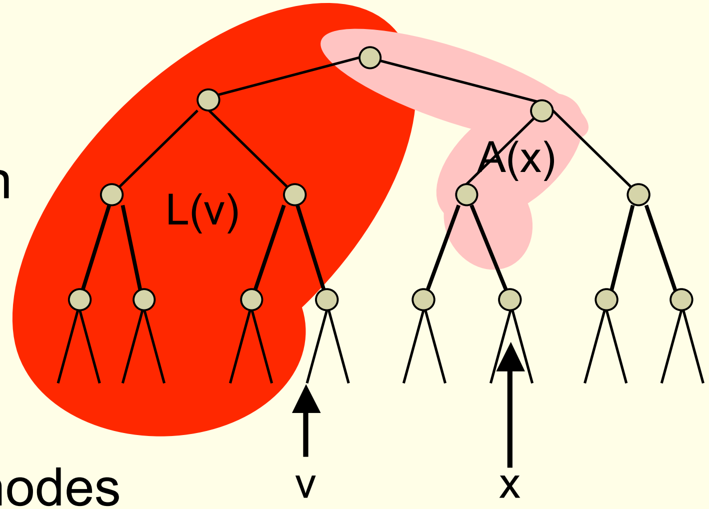
- ¾-quantile at time step N′ = 2N



- N′ = 2N, eN = e/2(2N)

Biased Quantiles [CKMS06]

- Domain-oriented [SBAS04]
  - Items drawn from [1…U]
  - Impose binary tree over domain
  - Want space to be O(log U)



- Maintain counts c_w on (subset of) nodes
  - Represents input items from subtree
  - L(v): counts to left of a leaf are certainly less
  - A(x): uncertainty in rank is from ancestors

Biased Quantiles: Results

- Maintain accuracy invariants
  - Deterministically bound ranks: $L(x) - A(x) \leq \text{rank}(x) \leq L(x)$
  - Bound possible ranks: $v \neq \text{lf}(v) \rightarrow C_v \leq (\varepsilon/\log U)\, L(v)$
  - Consequence: can find $r'(x)$ so $|r'(x) - \text{rank}(x)| \leq \varepsilon\, \text{rank}(x)$

- Results: can answer queries with error $\leq \varepsilon\, \text{rank}(x)$
  - Use space $O(1/\varepsilon \log(\varepsilon N) \log(U))$
  - Amortized update time $O(\log \log U)$
  - Lower bound on space of $O(1/\varepsilon \log(\varepsilon N))$

Stream Map

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

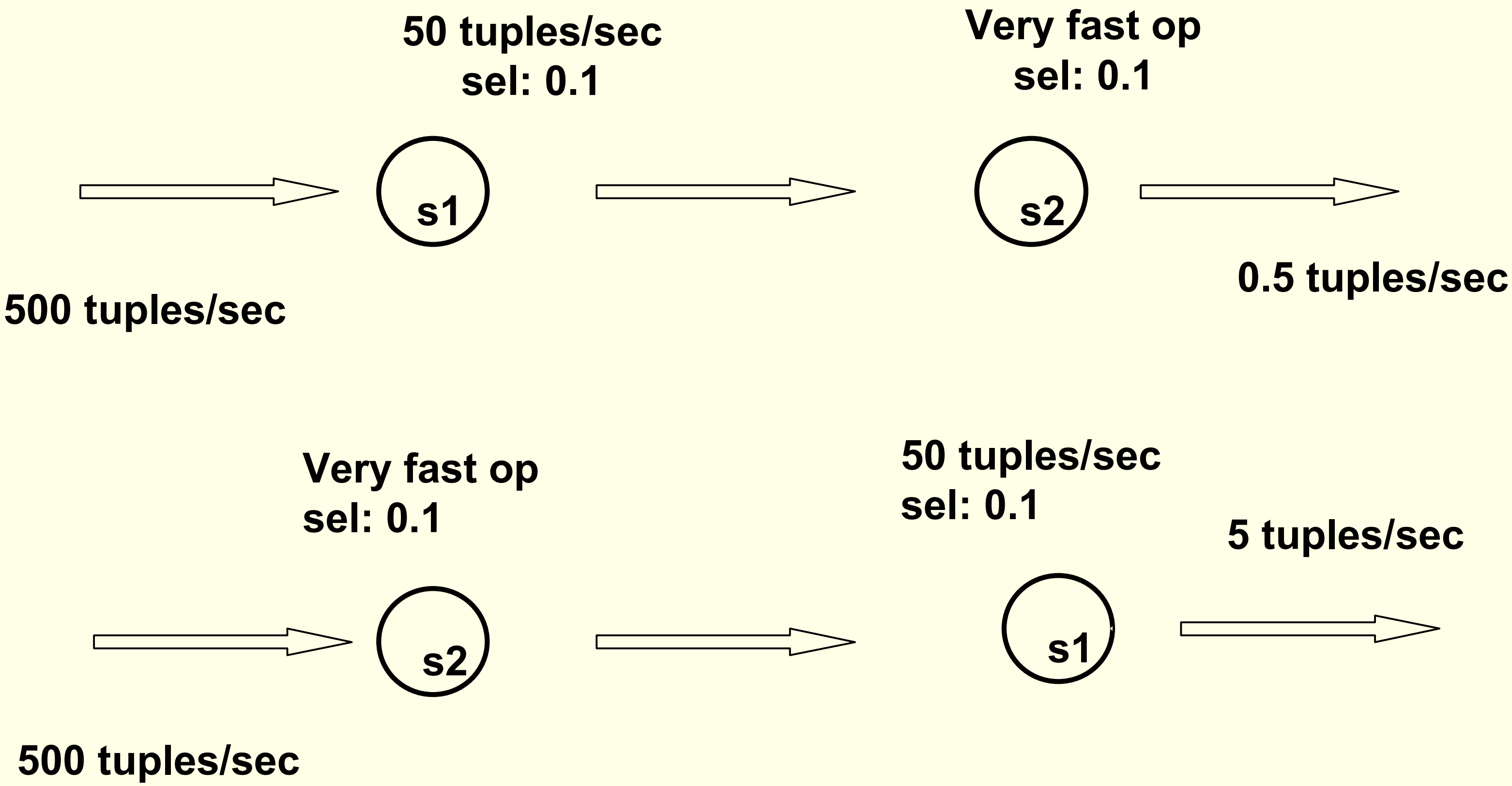
Optimization Objectives: Issues

- 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 rates of streams in query evaluation tree
  - 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

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: 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

Stream Map

- Part I: Motivation
- Part II: Query processing
  - Stream query language issues
  - Query operators
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: Gigascope DSMS

Multi-query Processing on Streams

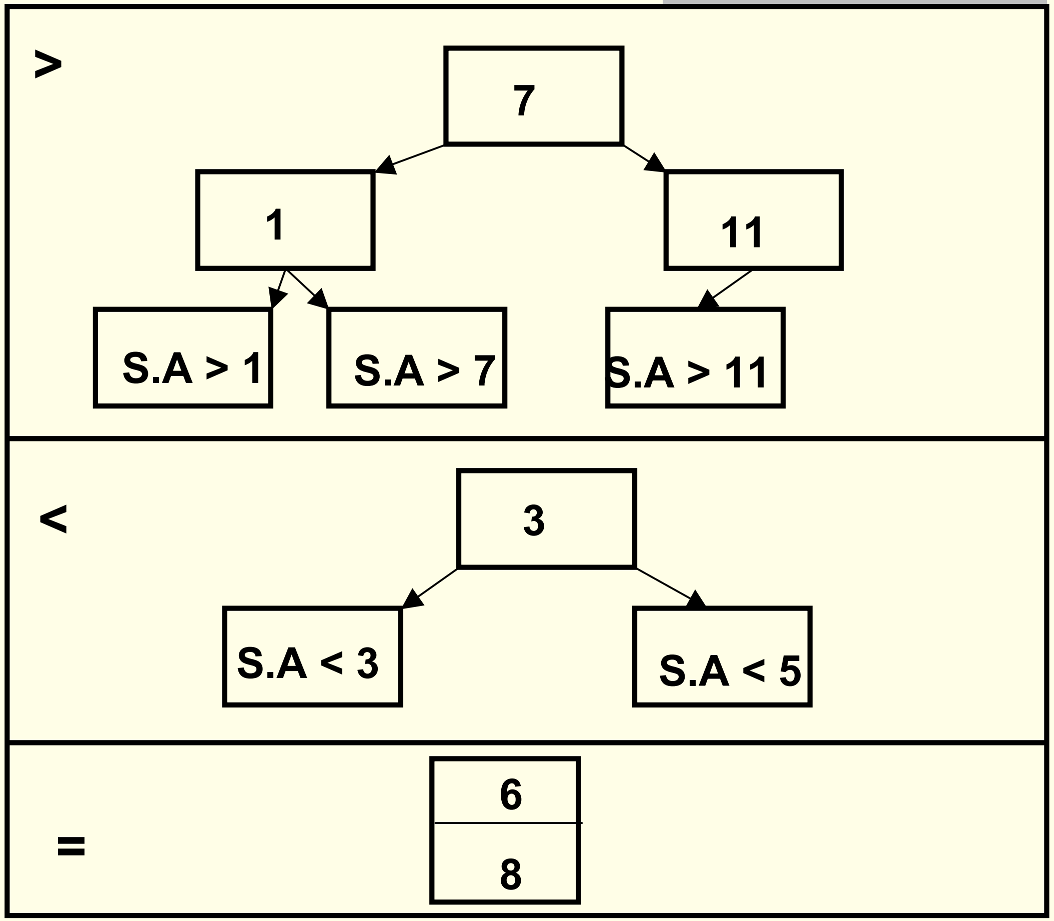
- In traditional multi-query optimization:
  - Result sharing among queries leads to better performance

- Similar issues arise when processing queries on streams:
  - Sharing between select/project expressions
  - Sharing between sliding window join expressions

Grouped Filters [MSHR02]

Select Predicates for Stream S.A
S.A > 1 S.A > 7 S.A > 11
S.A < 3 S.A < 5
S.A = 6 S.A = 8

Tuple S.A = 8



Shared Window Joins [HFAE03]

- Consider the two queries:

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

select count (distinct A.sourceIP)
from TCP A [window 1 min], TCP 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 Window Aggregates [AW04]

- Great opportunity for optimization as windows are highly shared

- Sliding window aggregates
  - Various aggregation functions (e.g., distributive, algebraic)
  - Various window types (time, tuple based)
  - Input models (single, multiple streams)

Stream Map

- Part I: Motivation
- Part II: Query processing
  - Stream query language issues
  - Query operators
  - Optimization objectives
  - Multi-query execution
  - Prototype systems
- Part III: Gigascope DSMS

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]
- …

Related DSMS Technologies

System	Data Stream Architecture	Data Model	Query Language	Query Answers	Query Plan
Aurora StreamBase	low-level	RS-in RS-out	Operators	approximate	QoS-based, load shedding
Gigascope	two level (low, high)	S-in S-out	GSQL	approximate	decomposition, distribution
Hancock	high-level	RS-in R-out	Procedural	exact, signatures	optimize for I/O, process blocks
Nile	high level	RS-in RS-out	SQL-based	approximate	incremental evaluation, multi-query
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

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

- 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
- 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
- Various aspects of optimized multi-query stream processing

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
  - No query language: queries specified in predicate calculus

Stream Map

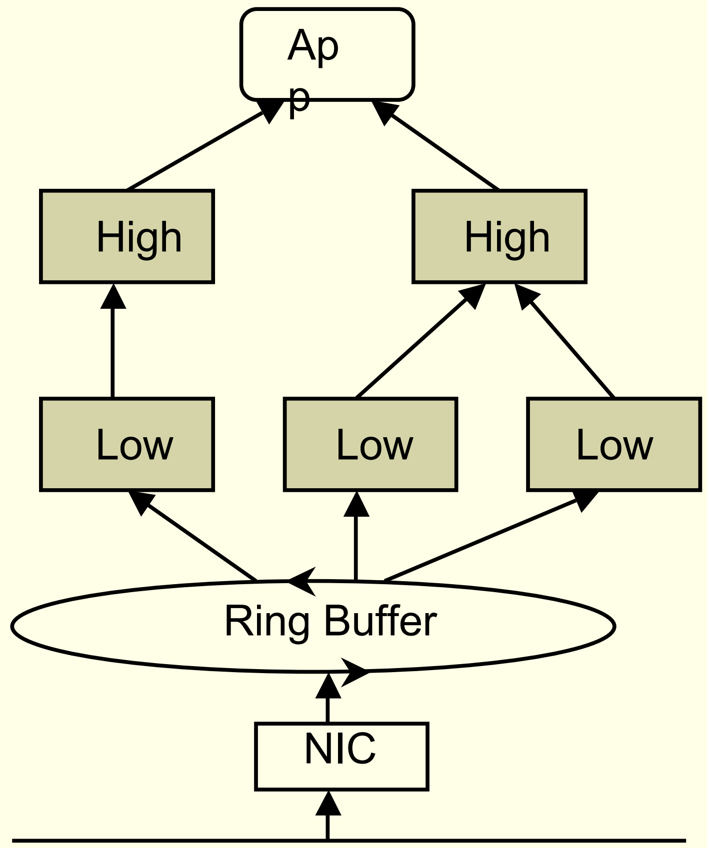
- Part I: Motivation
- Part II: Query processing
- Part III: Gigascope DSMS
  - Scalable aggregate query processing
  - Open Issues

Gigascope: Scalability

- Gigascope is a fast, flexible data stream management system
  - High performance at OC768 speeds (2 x 40 Gbit/sec)
  - Non-trivial queries at 200,000 pkts/sec using 38% of 1 CPU
- Monitoring platform of choice for AT&T IP network
- Scalability mechanisms
  - Two-level architecture: Query splitting, pre-aggregation
  - Distribution architecture: Query-aware stream splitting
  - Unblocking: Reduce data buffering
  - Sampling algorithms: Data reduction

Gigascope: Two-Level Architecture

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



Gigascope: Query Splitting

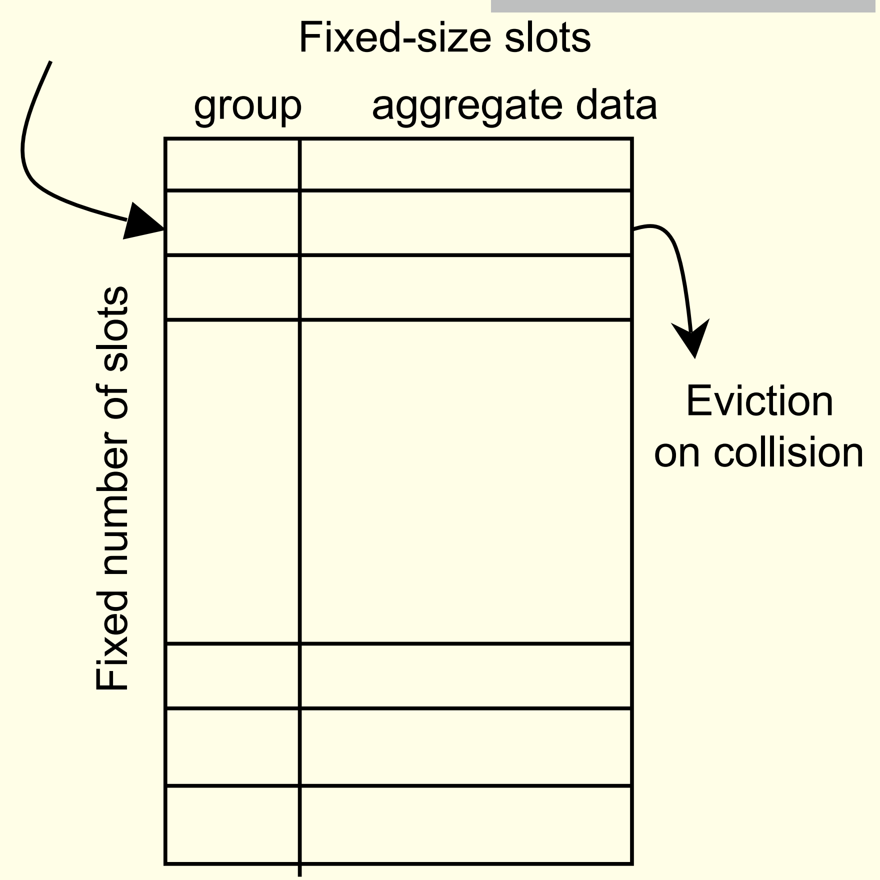
```
define { query_name smtp; }
select tb, destIP, sum(len)
from TCP
where protocol = 6 and
    destPort = 25
group by time/60 as tb, destIP
having count(*) > 1
```

```
select tb, destIP, sum(sumLen)
from SubQ
group by tb, destIP
having sum(cnt) > 1
```

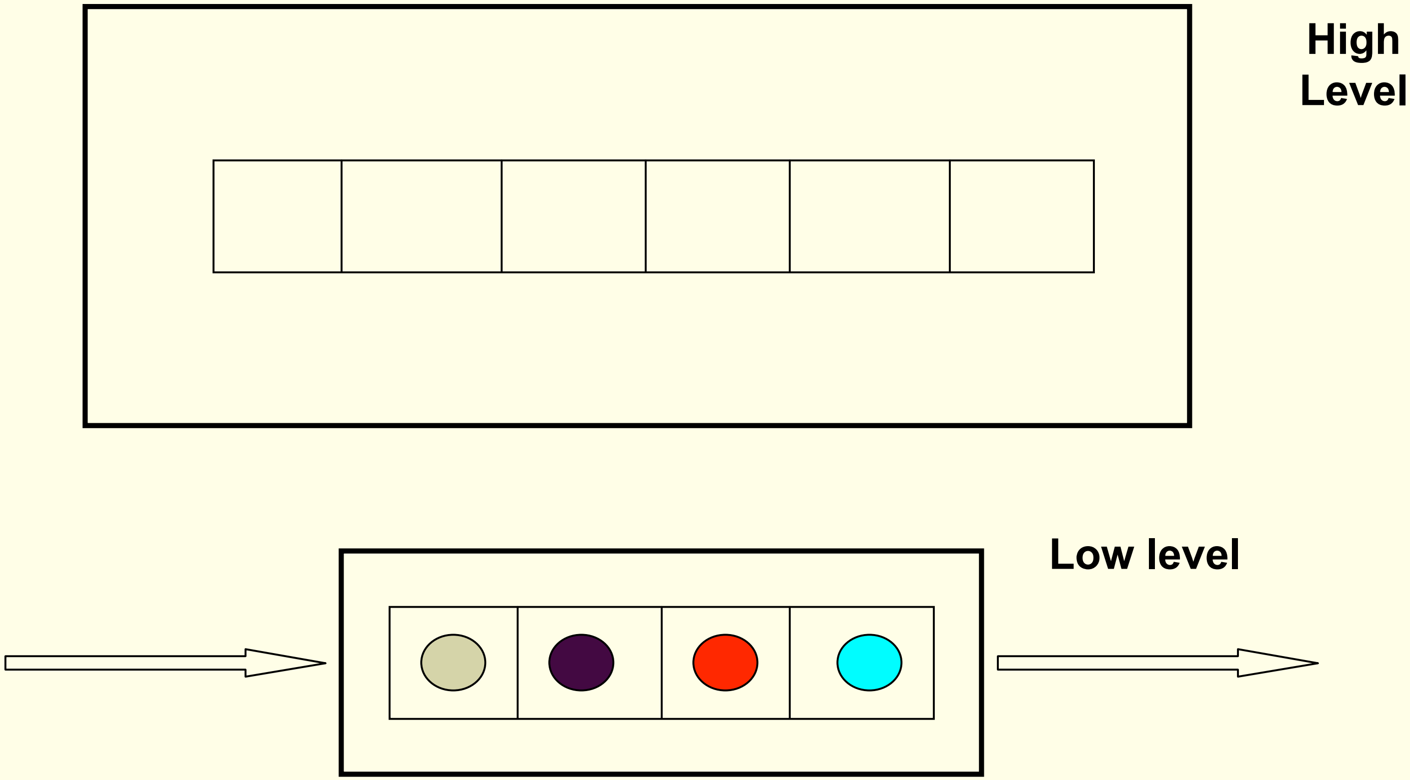
```
define { query_name SubQ; }
select tb, destIP, sum(len) as
    sumLen, count(*) as cnt
from TCP
where protocol = 6 and
    destPort = 25
group by time/60 as tb, destIP
```

Gigascope: Low-Level Aggregation

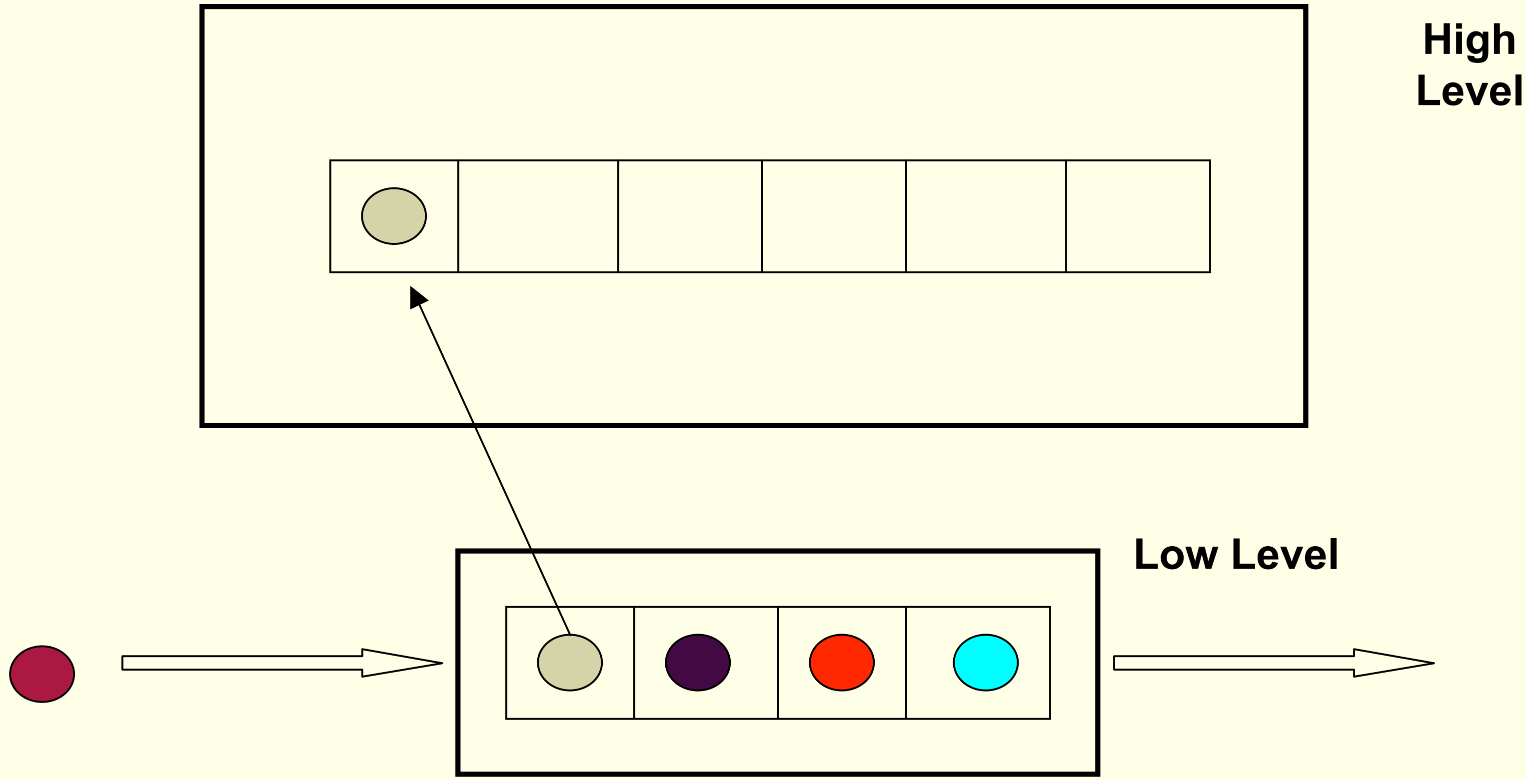
- Fixed number of slots for groups, fixed size slot for each group
- Direct-mapped hashing
- Optimizations
  - Limited hash chaining reduces eviction rate
  - Slow eviction of groups when epoch changes



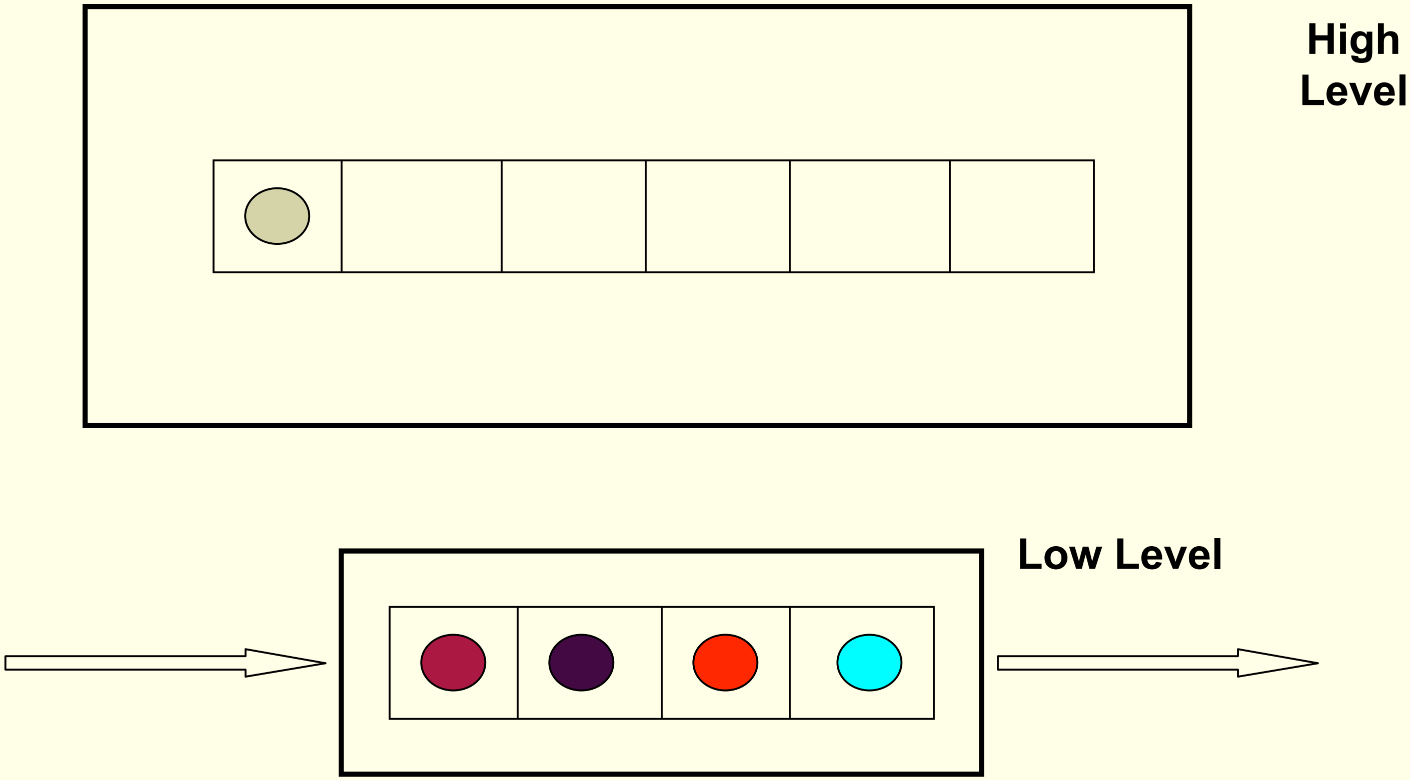
Aggregation in Gigascope



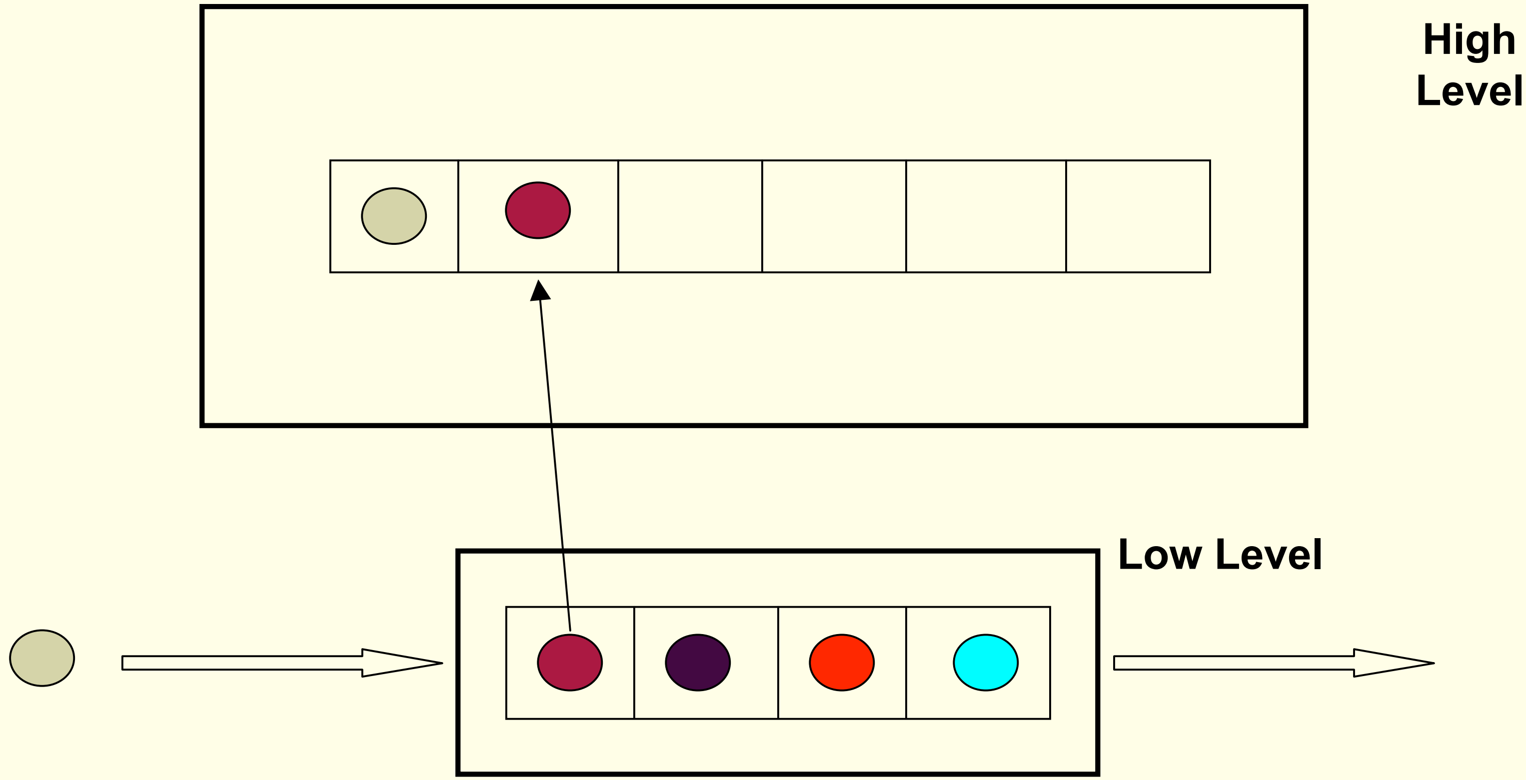
Aggregation in Gigascope



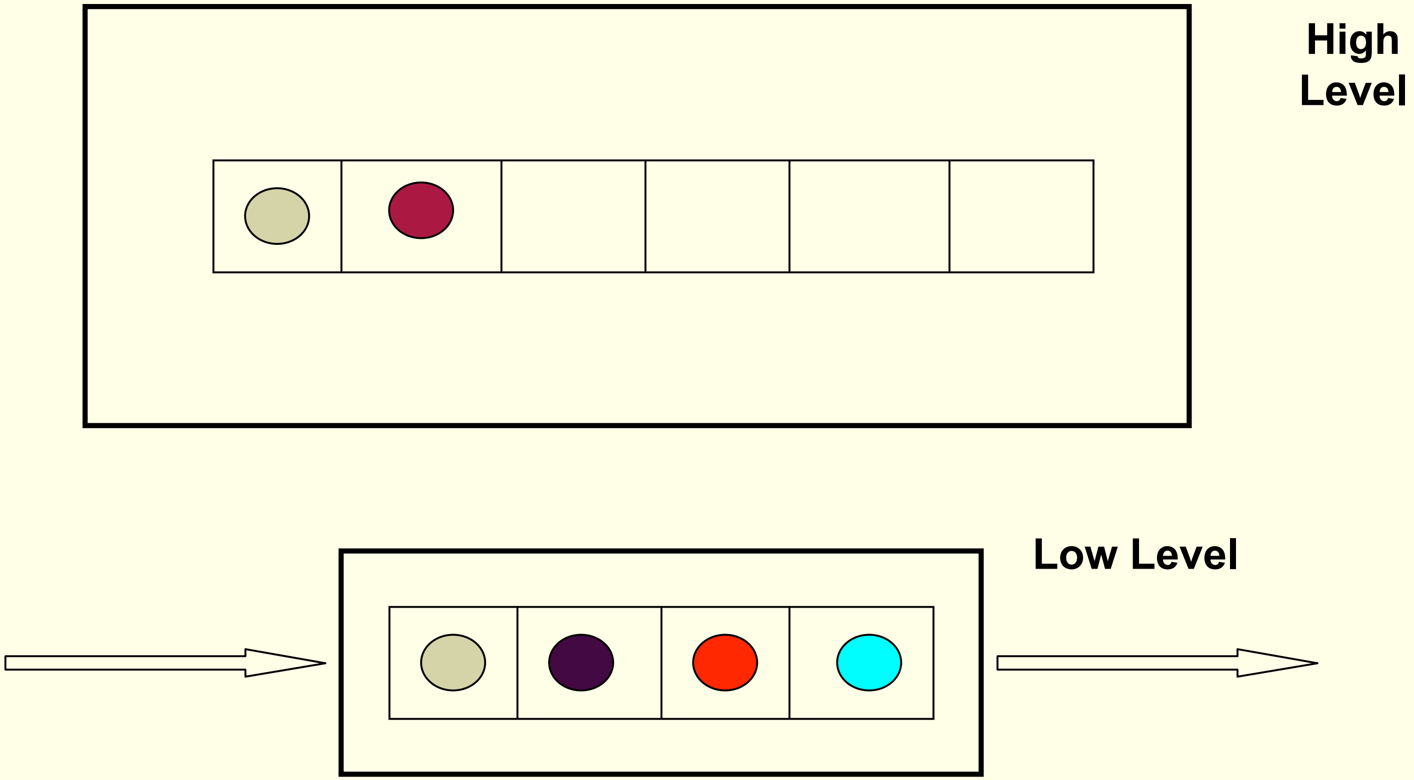
Aggregation in Gigascope



Aggregation in Gigascope



Aggregation in Gigascope



Gigascope: UDAF Specification

- Standard database UDAF: INIT, ITERATE, TERMINATE
- Gigascope UDAF: similar to standard database UDAF, but
  - Break TERMINATE into OUTPUT and DESTROY: enables, e.g., quantile(len, 0.9), quantile(len, 0.95), quantile(len, 0.99)
- Can support arbitrary data stream algorithms as UDAFs
  - GK quantile summary, CKMS (biased) quantile summary
  - Count-min (CM) sketch

Gigascope: UDAF Design Issues

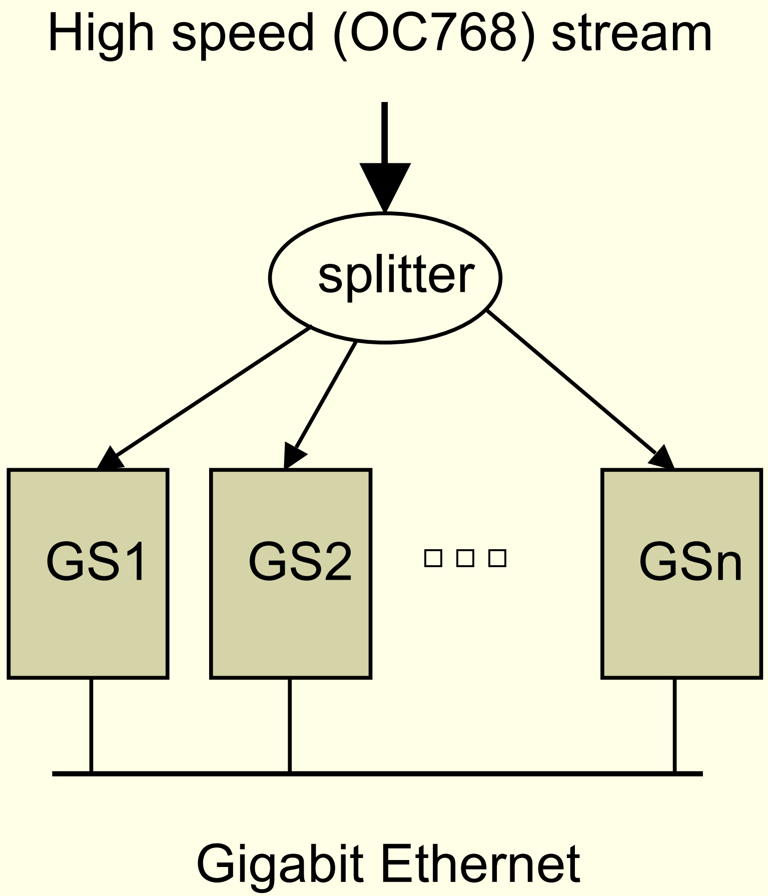
- Split processing effort between high and low level
- Processing at low-level saves processing at high-level
  - Data reduction, fewer transfers, fewer merges, etc.
- Too much processing at low-level causes packet drops
  - Quick-and-dirty filtering and aggregation
- Need to strike the right balance
  - Lightweight data structures, especially at low level
  - Avoid excessive processing at bottlenecks

Gigascope: Performance

Query	Low	High	Packets/sec
counting only	8%	0%	145,000
grouping aggregation	12.6%	0.5%	145,000
inverse distribution	25%	15.5%	142,000
UDAF	30%	43%	141,000
DDoS (join)	16.9%	3.1%	142,000
P2P (content)	10.7%	0%	139,000

Distributed Gigascope

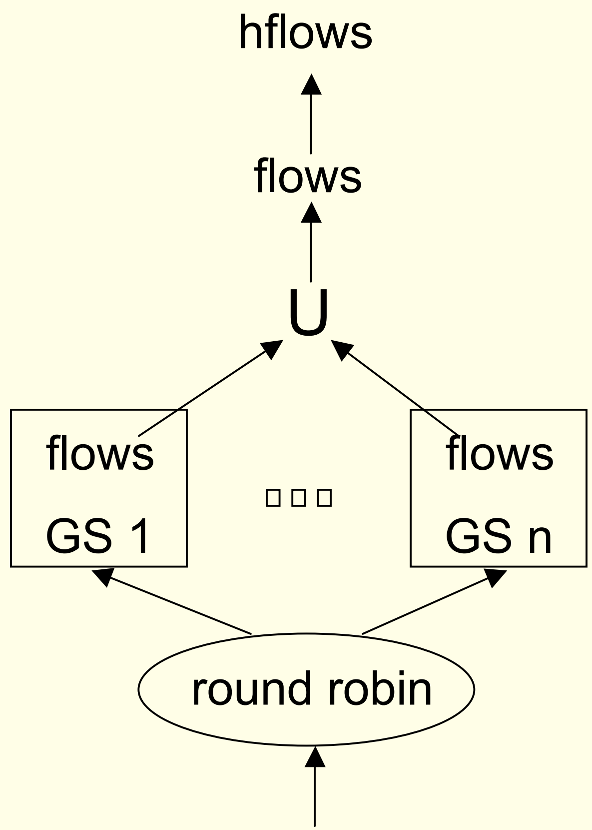
- Problem: OC768 monitoring needs more than one CPU
  - 2x40 Gb/s = 16M pkts/s
- Solution: split data stream, process query, recombine partitioned query results
- For linear scaling, splitting needs to be query-aware



Gigascope: Query-Unaware Splitting

define { query_name flows; }
select tb, srcIP, destIP,
    count(*)
from TCP
group by time/60 as tb, srcIP,
    destIP

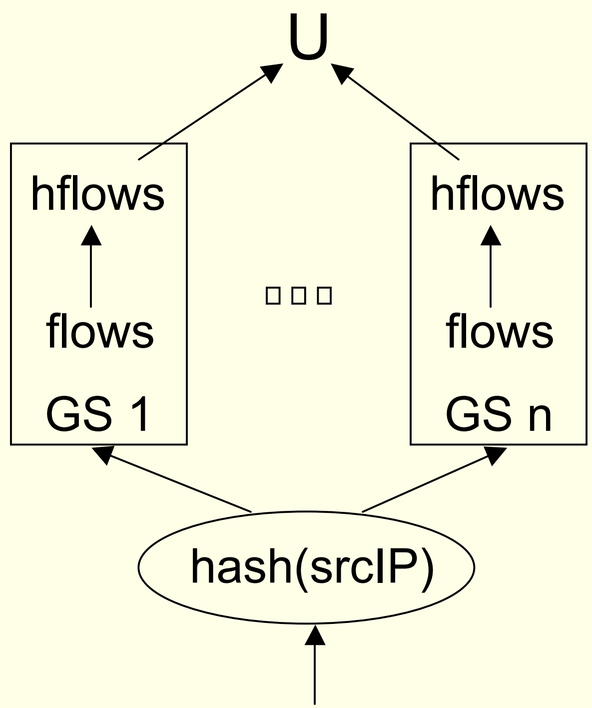
define { query_name hflows; }
select tb, srcIP, max(cnt)
from flows
group by tb, srcIP



Gigascope: Query-Aware Splitting

<u>define</u> { query_name flows; }
<u>select</u> tb, srcIP, destIP, count(*)
<u>from</u> TCP
<u>group by</u> time/60 as tb, srcIP, destIP

<u>define</u> { query_name hflows; }
<u>select</u> tb, srcIP, max(cnt)
<u>from</u> flows
<u>group by</u> tb, srcIP



Gigascope: Unblocking

- Issues
  - Produce useful output over potentially infinite streams
  - A link failure can stall an input stream
- Solution technique: Timestamps
  - Identify fields behaving like timestamps (monotone)
  - Determine tuple locality by query analysis on references
- Solution technique: Punctuation carrying “heartbeats”
  - Inject heartbeats into streams, propagate through query dag
  - Significant reduction in memory usage with low CPU cost

Gigascope: Sampling Algorithms

- Issues
  - Need sampling to deal with high volume streams (attacks)
- Solution technique: Single operator that can be specialized
  - Simple communication structure between samples, summary
  - Efficient implementation using multiple hash tables
- Solution technique: User-defined aggregate functions (UDAFs)
  - Separate UDAFs for distinct sampling algorithms
  - Added flexibility permits inter-sample communication

Stream Map

- Part I: Motivation
- Part II: Query processing
- Part III: Gigascope DSMS
  - Scalable aggregate query processing
  - Open Issues

Challenges and Opportunities

- Challenges
  - Large query sets: 100s of GSQL queries, black-box UDAFs
  - Data quality: inadequate understanding of network protocols
  - Network speeds increasing: OC48 → OC192 → OC768
- Opportunities
  - Multi-query optimization: predicates, joins, UDAFs, etc.
  - Stream integrity: PAC constraints, etc.
  - Using specialized hardware: GPUs, FPGAs, etc.

Multi-Query Optimization

- Challenge
  - 100s of GSQL queries, black-box UDAFs
- Traditional MQO problem: predicates, aggregates, joins, etc.
  - Fast identification of queries relevant to a record
- Novel MQO problem: optimizable, shareable UDAFs
  - Example: GSQL queries using different sampling strategies
  - Declarative characterization (specification?) of UDAFs

Stream Integrity

- Challenge
  - Complex protocols, inadequate understanding in practice
- Queries can return inexplicable results
  - Unlike in a DBMS, cannot go back to explore the raw data
- Need to formally characterize and monitor query pre-conditions
  - Example: stream sorted on time? multiple SYN packets?
  - PAC constraints to approximately quantify violations

Using Specialized Hardware

- Challenge
  - Network speeds increasing: OC48 → OC192 → OC768
- Using commodity hardware
  - GPUs for highly parallel computations with spatial locality
- Using specialized hardware
  - FPGAs to parse TCP packet headers
  - RegEx matchers to access application-level (HTTP) fields

Conclusions

- Data stream query processing has real applications
  - Need for sophisticated near-real time queries
  - Massive data volumes of transactions and measurements
- Gigascope is a flexible DSMS, used in practice
  - Designed to support complex aggregation on fast streams
  - Careful algorithm engineering essential for performance
- Wealth of challenging technical and practical problems exist
  - Resource limitations exist, especially at low-level
  - Important to think of the end-to-end architecture

Acknowledgements

- Colleagues
  - Graham Cormode, Lukasz Golab, Ted Johnson, Flip Korn, Nick Koudas, S. Muthukrishnan, Irina Rozenbaum, Vlad Shkapenyuk, Oliver Spatscheck
- Papers and tutorials
  - Data stream query processing tutorials at VLDB'03, ICDE'05
  - Papers in SIGMOD'03, VLDB'03, SIGMOD'04, ICDE'05, SIGMOD'05, DBSec'05, VLDB'05, PODS'06

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

References

- [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

References

- [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

References

- [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

References

- [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

References

- [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

References

- [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)

References

- [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