Streams Meeting, March 20, 2002.
@ Stanford

Stanford:

Jennifer Widom	widom@db.stanford.edu
Rajeev Motwani	rajeev@cs.stanford.edu
Jeff Ullman	ullman@cs.stanford.edu
Arvind Arasu	arvinda@cs.stanford.edu
Brian Babcock	babcock@cs.stanford.edu
Shivnath Babu	shivnath@cs.stanford.edu
Prasanna Ganesan	prasanna.ganesan@cs.stanford.edu
Gurmeet Manku	manku@cs.stanford.edu
Liadan O'Callaghan	loc@cs.stanford.edu
Rohit Varma	rohit.varma@cs.stanford.edu

Berkeley:

Mike Franklin	franklin@cs.berkeley.edu
Sirish Chandrasekaran	sirish@cs.berkeley.edu
Yanlei Diao	diaoyl@cs.berkeley.edu
Sam Madden	madden@cs.berkeley.edu

OGI:

Dave Maier	maier@cse.ogi.edu
Pete Tucker	ptucker@cse.ogi.edu

Dave Meier:

Pipelined Evaluation
- Requires some local state at operators

Monotonic Evaluation:
- Basic SPJ, dupl.. elim are monotonic
- Is Nest (aka groupby) monotonic?
- What does monotonic mean?
- Maier: If a tuple's in the answer it stays in the answer forever.
- Widow: If the input gets bigger, the answer gets bigger.
- Groups don't grow monotonically in Widom's view – may change as new values arrive

In Maier's view, montonicity of nest depends on definition of "less-than" (e.g. ordering) – if ordering is subset, not nest not monotonic, if substructure, nest is monotonic.

One way to do it is to make "reconstitution function" handle it – e.g. make client compute groups. Jennifer sez reconstitution function is a lot like deltas. Maier wants fancy reconstitution functions (e.g. partial preaggregation.) Zaniello talks about making aggregates monotonic.

Maier sez you need to "describe the intent of the stream" – is it deltas, values, groups, partial groups?

Maier: Jayaval / Kristen showed that streaming aggregates (e.g. nonmonotonic) results might make deltas preferable.

Maier: Kinds of stream semantics:

- Append
- Delta
- Merge
- "Broadcast Streams" – a sequence of snapshots, cyclic updates with modified entries in the cycle.
- Strict snapshots?
- Invalidations

What does merge mean for two XML documents?

Two streams; which is the correct merge?

```
                                              D
A – B – D                                    /
                         D                  B
                        /                  /
          A – B                           A
                        \                  \
                         C                  B
                                             \
                                              C
A – B – C
```

OGI wants to do it via templates. DTDs don't specify this in enough detail, Xschema probably doesn't either.

<u>Pete Tucker:</u>

Punctuated Streams
Deal with both unbounded state and blocking operators

Punctuations terminate subsets within the streams; e.g. end of time windows, groups, etc. Allows state to be discarded early, and allow partial results to be output.

Punctuation semantics must be understood by operators. For now, punctuations are very simple – terminate subsets, apply only to a single attribute. But could imagine lots of more complicated punctuations.

Used so far:

Seen everything in a range. (or up to a value)
Seen everything in a list
Seen all occurrences of a value
Seen everything (e.g EOAttr)

Can punctuations have a "lease" – e.g. valid for a certain time range?

For now, punctuations are idempotent. Stream is "grammatical" – always ok to send punctuations later, or to periodically send cumulative punctuations. Not clear what this means for more complicated punctuations.

Three kinds of rules:

Pass rules: Indicates what a blocking operator output when its sees a punctuation.
Purge rules: What can a stateful operator discard when it sees punctuation?
Propagation rules: When can a operator pass a punctuation?

Sirish: What about pre-punctuation that indicates what the user can expect to see?
Maier: That'd be a different kind of stream semantics – useful.

Current state: partially implemented in Niagara Query Engine.

Are punctuations and windows and epochs the same thing?
Sam: windows are essentially "implicit" punctuations
Jennifer: different kinds of constraints on streams:

- *Schema level* stream constraints (e.g. ordering, or strict referential integrity)
    - Allow data to be thrown out
- *Data-level* constraints (e.g. punctuations)
- *Query-level* constraints (e.g. time windows)
- *Operator-level* constraints

Are constraints always about optimization? Schema level, and data level are, query level maybe (system constraints are about optimization, user constraints are about query correctness). (Yarick Geryz – Last year's SIGMOD industrial track – inexact properties.)

Correlated aggregates (Gehrke), last years PODS (or SIGMOD), talks about "landmarks", which look a lot like punctuations.

Chronicle data model – any join is an implicit band join on time, which constrains things.

Ullman – seems like you should be able to reason about the types of punctuations that can be output, demonstrate that state actually gets purged everywhere, etc. At least for the very restricted set of punctuations currently being considered.

<u>Jennifer</u>

Data Stream Queries

- Answer availability
    - One time
    - Multiple time
    - Continuous ("standing")
- Registration time
    - Predefined
    - Ad Hoc
- Stream Access
    - Arbitrary
    - Sliding Window (special case size = 1)

Applications
- Network management and traffic engineering
    - Streams of measurements and packet traces
- Network security
    - Network packet streams, user session information
    - Queries: URL filtering, intrusion detection DOS attacks, viruses
- Financial Applications
    - Streams of trading data, stock tickers, news feeds
    - Queries: arbitrage opportunities, analytics, patterns
- Web track and personalization
- Massive Databases (e.g. astronomy, physics data)

Query Evaluation: Distributed Streams
- Many physical streams but one logical stream (not physically combining data!)
    - e.g yahoo top 100 pages
- Correlate streams at distributed servers
    - e.g. network monitoring
- Many streams controlled by a few servers
    - e.g. sensor networks in a centralized environment

Issues / concepts:
move processing to streams, not streams to processor
approximation-bandwidth tradeoff

Architecture:

- Operators have synopses; more memory == bigger synopses
- Synopses orthogonal to operators; operators can use different synopses
- Operators in query plan, connected by queries, scheduled by scheduler
- Approximation inherent: e.g. duplicate elimination – more memory improves the quality of the results (fewer duplicates)
- Hope to share synopses – don't know how.
- Global memory allocation problem: lower operator with a small synopses limits amount of useful state at higher level.
- Query plans == Fjords!

DSMS Internals
Operators, synopses, queues
Accuracy vs. memory tradeoff in every operator
Every operator adapts gracefully irrespective of memory
Scheduler: round robin scheduler

System Implementation
"Developers workbench":
- Submit queries in extended SQL or algebra
- Submit or edit query plans in XML or GUI
- Query plan execution visualizer
- On-the-fly modification of memory allocation, scheduling policies, etc.

Lots of algorithmic work on synopses (published stuff via Rajeev & co.)

Constraints:
- Important to exploit constraints in query processing (specified at schema level)
    - Foreign key joins
    - Referential integrity : combine streams
    - Clustering, ordering : what tuples are in time window
- Need not be exact (e.g. k-clustered)
- Reduce memory requirements
- Unblock blocking operators (e.g. if you know a stream is sorted on time, you can output results up to the most recent time received)

Some relationship to punctuations – seems as though some punctuations can be expressed as schema level properties…

Approximation in query processing

- Understanding behavior of approximate operators when composed
- Memory allocation to operators in a plan, given per operator memory-accuracy curve
- Best query plan, assuming best memory allocation
- Multiple (weighted) queries sharing resources

Operator tells you how it's approximating, maybe outputs confidence intervals. If we understand how approximate operators compose, how do I allocate memory globally to maximize accuracy? Then, how do I choose the best query plan, assuming best memory allocation? Finally, how to I allocate memory between multiple queries or multiple query plans.

Mike: What do accuracy curves look like? Don't really know
Maier: Could you do query checkpoints as XML? Jennifer: XML in DSMS is not about state storage – it's about a query plan, or memory allocation, but not actual tuples.

Answers tend to get less accurate as they flow up the query plan in this scheme. Lower level operators could make use of state stored in higher level operators to decide what to discard / keep.

Adaptive to resource allocation / memory, not adaptivity of query plans. Not even really attacking problem of query plan generation – in initial implementation users implement just query graphs.

<u>Sam and Sirish</u>
Tuples are all timestamped.

Windows are based on timestamps. (also can specify by # of tuples? or punc?)

Tuple timestamps are applied either when the data is created
in the source, or when the tuples arrive.

Time series analysis vs. most current
    reconstitution - two functions: one to time series, one to most recent

<u>Yanlei</u>

Maier: What is sent on to the person when a match is found? Whole document
or some subset or the matching path? XQuery has a construction phase -- once you
know a document matches, you have to build the result. Then the result
has to be delivered to the user. Delivery seems to be the bottleneck -- probably
need distributed routing / delivery.

Maier: Is it necessary to find all matches or just one match? Need to support both. NFA supports both. Can we optimize in the case where you just want to find one? Not really -- don't prune the NFA to eliminate paths you've already navigated.

XML parsing is bottleneck! Not filter cost. Result collection is more expensive than filter too, in YFilter (not Xfilter or hybrid scheme (Rajeev Rastogi?).)

Maier: Parsing is a problem in their world too. Found that combining 10 documents into a single document yielded a significant performance benefit. Theory: symbol table construction is important.

Yanlei: Java parsers 60-80ms, C parsers 6-10 ms.

Maier: This is a stream shredder? Yup.

Jennifer: This really looks like a query plan. Yup.

Delivery seems to dominate result construction.

Open Questions:

Streams benchmark?
Streaming data model? Can we converge on a data / query model we agree on?

Benchmark might be a good starting point -- we'll have to agree what queries to run and what data to run them over.

Stream Taxonomy

	Sensors	ClickStream	Networking Monitoring and Routers	Call Record Stuff	Financial	Baseball (Event Streams)	XML dissemination Pub/Sub
# Of Clients	1-1000	100	1	2	10^7		10^6
Delivery Requirements							
Content Complexity	low	low (homog)	med	low	low	med (hetero)	high
Query Types	relational, time-series, alerts	data-mining	relational, time-series, data-mining, alerts	data-mining	alerts, time series	xquery, aggregates	filtering, alerts, excerpting, restructuring

							(not aggregating /combining)
Distribution							
Stream Semantics						stream of xml fragments	stream of xml
Data Arrival Rate	low-high	med	very high	high	med	low-med	med
Data Delivery	both	push	push	push	both	both	push
Query Flux	?	low	low	low	?	?	?
Query Load	?	med	high	low	high	?	high
Accuracy Requirements	low; variable	low	med?	low (load), high (fraud)	high	?	high
Lossiness	yes	no	no	no	no	no	no
rate flux							

<u>Glossary</u>

Band Join
Sliding Window Join
Window Join
Active/Inactive Queries
Transfer Queries: Logging a view
Predefined == queries exist before data
Ad-hoc == streams exist before queries
Flying join