Advanced Topics in Distributed Systems

Data Stream Management Systems

- Applications, Concepts, and Systems –

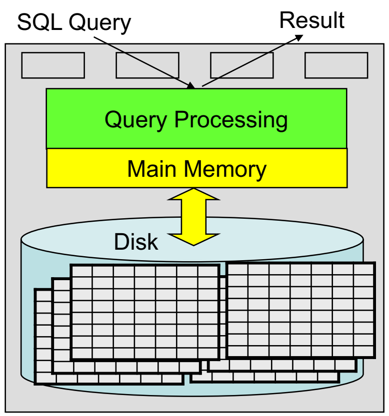
Vera Goebel & Thomas Plagemann
University of Oslo

DSMS – Part 1 & 2

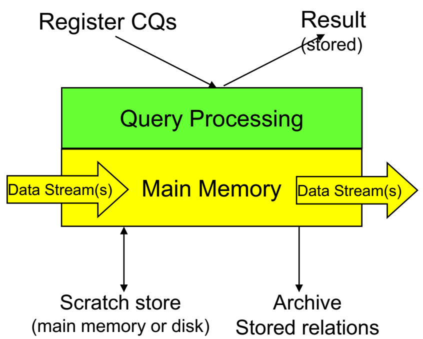
- Introduction:
  - What are DSMS? (terms)
  - DSMS vs. DBMS
  - Why do we need DSMS? (applications)
- Concepts and issues:
  - Architecture(s)
  - Data modeling
  - Query processing and optimization
  - Data Reduction & Stream Mining
- Example 1:
  - Network monitoring with TelegraphCQ, STREAM, and Borealis
- Example 2:
  - DSMS for sensor networks (Aurora)
- Summary: Open issues & conclusions

Handle Data Streams in DBS?

Traditional DBS



DSMS



Data Management

- Traditional DBS:
  - stored sets of relatively static records with no pre-defined notion of time
  - good for applications that require persistent data storage and complex querying
- DSMS:
  - support on-line analysis of rapidly changing data streams
  - *data stream*: real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items, too large to store entirely, not ending
  - continuous queries

Data Management:
Comparison - DBS versus DSMS

Database Systems (DBS)	DSMS
Persistent relations (relatively static, stored)	Transient streams (on-line analysis)
One-time queries	Continuous queries (CQs)
Random access	Sequential access
“Unbounded” disk store	Bounded main memory
Only current state matters	Historical data is important
No real-time services	Real-time requirements
Relatively low update rate	Possibly multi-GB arrival rate
Data at any granularity	Data at fine granularity
Assume precise data	Data stale/imprecise
Access plan determined by query processor, physical DB design	Unpredictable/variable data arrival and characteristics

Adapted from [Motawani: PODS tutorial]

Related DBS Technologies

- Continuous queries
- Active DBS (triggers)
- Real-time DBS
- Adaptive, on-line, partial results
- View management (materialized views)
- Sequence/temporal/timeseries DBS
- Main memory DBS
- Distributed DBS
- Parallel DBS
- Pub/sub systems
- Filtering systems
- …

=> Must be adapted for DSMS!

DSMS Applications

- Sensor Networks:
  - Monitoring of sensor data from many sources, complex filtering, activation of alarms, aggregation and joins over single or multiple streams
- Network Traffic Analysis:
  - Analyzing Internet traffic in near real-time to compute traffic statistics and detect critical conditions
- Financial Tickers:
  - On-line analysis of stock prices, discover correlations, identify trends
- On-line auctions
- Transaction Log Analysis, e.g., Web, telephone calls, …

Data Streams - Terms

- 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
  - Sensor networks: physical phenomena, road traffic
  - IP network: traffic at router interfaces
  - Earth climate: temperature, moisture at weather stations

VLDB 2003 Tutorial [Koudas & Srivastava 2003]

Motivation for DSMS

- Large amounts of interesting data:
  - deploy transactional data observation points, e.g.,
    - AT&T long-distance: ~300M call tuples/day
    - AT&T IP backbone: ~10B IP flows/day
  - generate automated, highly detailed measurements
    - NOAA: satellite-based measurement of earth geodetics
    - Sensor networks: huge number of measurement points

VLDB 2003 Tutorial [Koudas & Srivastava 2003]

Motivation for DSMS (cont.)

- Near real-time queries/analyses
  - ISPs: controlling the service level
  - NOAA: tornado detection using weather radar data

- Traditional data feeds
  - Simple queries (e.g., value lookup) needed in real-time
  - Complex queries (e.g., trend analyses) performed off-line

Motivation for DSMS (cont.)

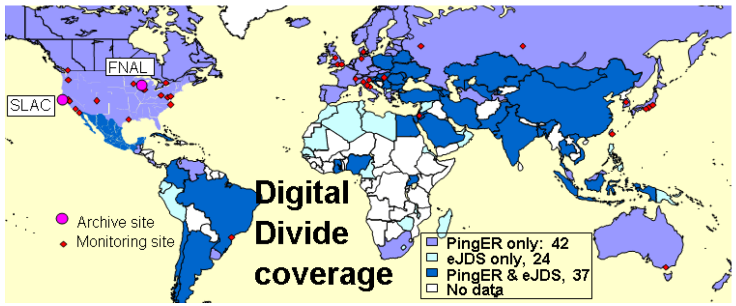
- Performance of disks:

	1987	2004	Increase
CPU Performance	1 MIPS	2,000,000 MIPS	2,000,000 x
Memory Size	16 Kbytes	32 Gbytes	2,000,000 x
Memory Performance	100 usec	2 nsec	50,000 x
Disc Drive Capacity	20 Mbytes	300 Gbytes	15,000 x
Disc Drive Performance	**60 msec**	**5.3 msec**	**11 x**

Source: Seagate Technology Paper: " Economies of Capacity and Speed: Choosing the most cost-effective disc drive size and RPM to meet IT requirements"

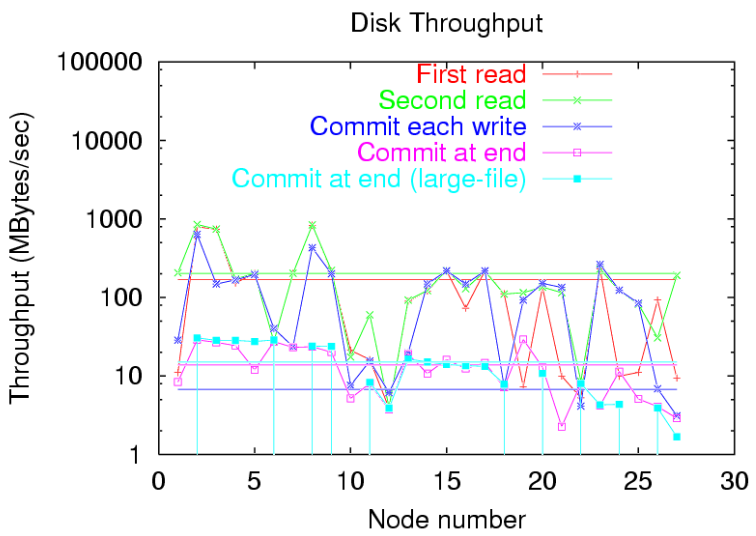
Motivation for DSMS (cont.)

- The PingER project:
  - Believed to be the most extensive Internet end-to-end performance monitoring tool in the world



Source: http://www-iepm.slac.stanford.edu/paperwork/index_2003.html

Motivation for DSMS (cont.)



Source: IEPM Project, http://www-iepm.slac.stanford.edu/bw/disk_res.html

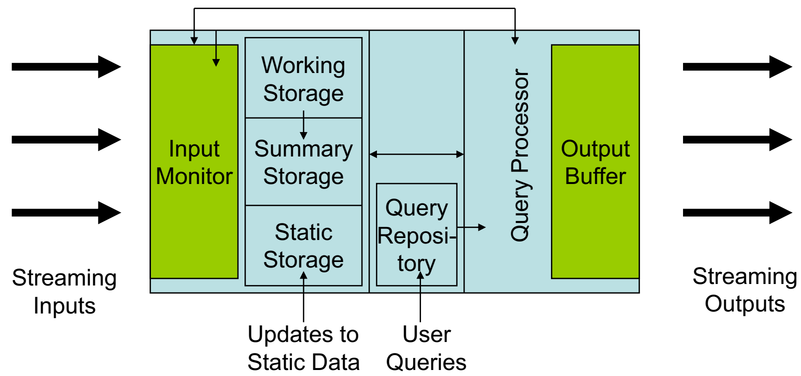
Motivation for DSMS (cont.)

- Take-away points:
  – Large amounts of raw data
  – Analysis needed as fast as possible
  – Data feed problem

Application Requirements

- Data model and query semantics: order- and time-based operations
  – Selection
  – Nested aggregation
  – Multiplexing and demultiplexing
  – Frequent item queries
  – Joins
  – Windowed queries
- Query processing:
  – Streaming query plans must use non-blocking operators
  – Only single-pass algorithms over data streams
- Data reduction: approximate summary structures
  – Synopses, digests => no exact answers
- Real-time reactions for monitoring applications => active mechanisms
- Long-running queries: variable system conditions
- Scalability: shared execution of many continuous queries, monitoring multiple streams
- Stream Mining

Generic DSMS Architecture



[Golab & Özsu 2003]

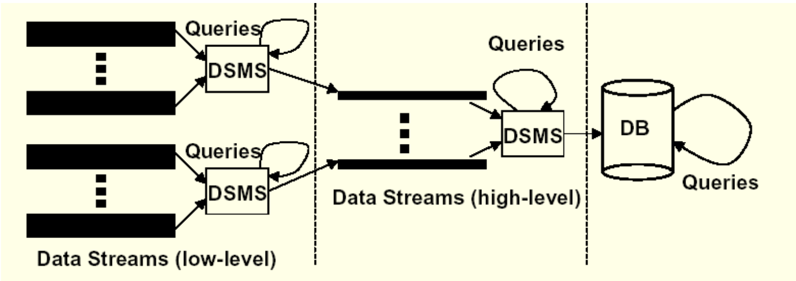
DSMS: 3-Level Architecture

DBS

- Data feeds to database can also be treated as data streams
- Resource (memory, disk, per-tuple computation) rich
- Useful to audit query results of DSMS
- Supports sophisticated query processing, analyses

DSMS

- DSMS at multiple observation points, (voluminous) streams-in, (data reduced) streams-out
- Resource (memory, per tuple computation) limited, esp. at low-level
- Reasonably complex, near real-time, query processing
- Identify what data to populate in DB



VLDB 2003 Tutorial [Koudas & Srivastava 2003]

Data Models

- Real-time data stream: sequence of data items that arrive in some order and may be seen only once.
- Stream items: like relational tuples
  - relation-based models, e.g., STREAM, TelegraphCQ; or instanciations of objects
  - object-based models, e.g., COUGAR, Tribeca
- Window models:
  - Direction of movement of the endpoints: fixed window, sliding window, landmark window
  - Physical / time-based windows versus logical / count-based windows
  - Update interval: eager (update for each new arriving tuple) versus lazy (batch processing -> jumping window), non-overlapping tumbling windows

Timestamps

- Explicit
  - Injected by data source
  - Models real-world event represented by tuple
  - Tuples may be out-of-order, but if near-ordered can reorder with small buffers
- Implicit
  - Introduced as special field by DSMS
  - Arrival time in system
  - Enables order-based querying and sliding windows
- Issues
  - Distributed streams?
  - Composite tuples created by DSMS?

Time

- Easiest: global system clock
  - Stream elements and relation updates timestamped on entry to system
- Application-defined time
  - Streams and relation updates contain application timestamps, may be out of order
  - Application generates “heartbeat”
    - Or deduce heartbeat from parameters: stream skew, scrambling, latency, and clock progress
  - Query results in application time

Queries - I

- DBS: one-time (transient) queries
- DSMS: continuous (persistent) queries
  - Support persistent and transient queries
  - Predefined and ad hoc queries (CQs)
  - Examples (persistent CQs):
    - Tapestry: content-based email, news filtering
    - OpenCQ, NiagaraCQ: monitor web sites
    - Chronicle: incremental view maintenance
- Unbounded memory requirements
- Blocking operators: window techniques
- Queries referencing past data

Queries - II

- DBS: (mostly) exact query answer
- DSMS: (mostly) approximate query answer
  - Approximate query answers have been studied:
    - Synopsis construction: histograms, sampling, sketches
    - Approximating query answers: using synopsis structures
    - Approximate joins: using windows to limit scope
    - Approximate aggregates: using synopsis structures
- Batch processing
- Data reduction: sampling, synopses, sketches, wavelets, histograms, …

One-pass Query Evaluation

- DBS:
  - Arbitrary data access
  - One/few pass algorithms have been studied:
    - Limited memory selection/sorting: *n*-pass quantiles
    - Tertiary memory databases: reordering execution
    - Complex aggregates: bounding number of passes
- DSMS:
  - Per-element processing: single pass to reduce drops
  - Block processing: multiple passes to optimize I/O cost

Query Plan

- DBS: fixed query plans optimized at beginning
- DSMS: adaptive query operators
  - Adaptive plans Adaptive query plans have been studied:
    - Query scrambling: wide-area data access
    - Eddies: volatile, unpredictable environments

Query Languages & Processing

- Stream query language issues (compositionality, windows)
- SQL-like proposals suitably extended for a stream environment:
  - Composable SQL operators
  - Queries reference relations or streams
  - Queries produce relations or streams
- Query operators (selection/projection, join, aggregation)
- Examples:
  - GSQL (Gigascope)
  - CQL (STREAM)
- Optimization objectives
- Multi-query execution

Query Languages

3 querying paradigms for streaming data:

1. Relation-based: SQL-like syntax and enhanced support for windows and ordering, e.g., CQL (STREAM), StreaQuel (TelegraphCQ), AQuery, GigaScope
2. Object-based: object-oriented stream modeling, classify stream elements according to type hierarchy, e.g., Tribeca, or model the sources as ADTs, e.g., COUGAR
3. Procedural: users specify the data flow, e.g., Aurora, users construct query plans via a graphical interface

(1) and (2) are declarative query languages, currently, the relation-based paradigm is mostly used.

Approximate Query Answering Methods

- Sliding windows
  - Only over sliding windows of *recent stream data*
  - Approximation but often more desirable in applications
- Batched processing, sampling and synopses
  - Batched if update is fast but computing is slow
    - Compute periodically, not very timely
  - Sampling if update is slow but computing is fast
    - Compute using sample data, but not good for joins, etc.
  - Synopsis data structures
    - Maintain a small *synopsis* or *sketch* of data
    - Good for querying historical data
- Blocking operators, e.g., sorting, avg, min, etc.
  - Blocking if unable to produce the first output until seeing the entire input

[Han 2004]

Query Optimization

- DBS: table based cardinalities used in query optimization
  => Problematic in a streaming environment
- Cost metrics and statistics: accuracy and reporting delay vs. memory usage, output rate, power usage
- Query optimization: query rewriting to minimize cost metric, adaptive query plans, due to changing processing time of operators, selectivity of predicates, and stream arrival rates
- Query optimization techniques
  - stream rate based
  - resource based
  - QoS based
- Continuously adaptive optimization
- Possibility that objectives cannot be met:
  - resource constraints
  - bursty arrivals under limited processing capability

Disorder in Data Streams

- Many queries over data streams rely on some kind of order on the input data items
  - Can often use more efficient operator implementations if the input is sorted on “interesting attributes” (e.g. aggregates)
- What causes disorder in streams?
  - Items from the same source may take different routes
  - Many sources with varying delays
  - May have been sorted on different attribute
- Sorting a stream may be undesirable
- May be more than one possible interesting order over a stream
  - For example, data items may have *creation time* and *arrival time*
  - Sorted on arrival time, but creation time also interesting

Punctuations

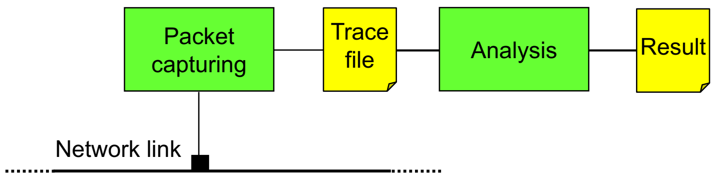
- Punctuations embedded in stream denote end of subset of data
  - Unblocks blocking operators
  - Reduces state required by stateful operators
- New operator: Punctuate
  - Has special knowledge regarding the input stream
    - timer-based, k-constraints, communication with stream source
  - Emits punctuations in source schema based on special knowledge
- Punctuations can help in two ways:
- Maintain order – Punctuations unblock sort
  - Similar to approach in Gigascope
  - Order-preserving operators include sort behavior for punctuations
- Allow disorder – Punctuations define the end of subsets
  - Operators use punctuations, not order, to output results
  - Reduces tuple latency

IP Network Application:
P2P Traffic Detection

- AT&T IP customer wanted to accurately monitor P2P traffic evolution within its network
  - Netflow can be used to determine P2P traffic volumes using TCP port number found in Netflow data
  - P2P traffic might not use known P2P port numbers
  - Using Gigascope SQL-based packet monitor
    - Search for P2P related keywords within each TCP datagram
    - Identified 3 times more traffic as P2P than Netflow
  - **Lessons**:
    - Essential to query massive volume data streams
    - Layer independence
    - Correlation of different sources (different app.)

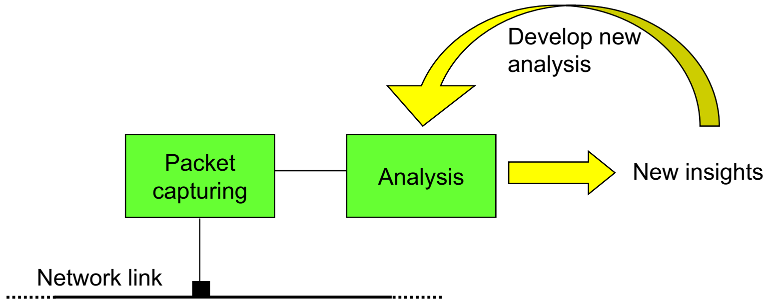
Example 1: Traffic Analysis

- Need to analyze Internet traffic is increasing ....
- .... and so is the number of tools for this
- Examples:
  - ISP monitor service levels, look for bottlenecks,etc.
  - development of new protocols, like P2P
- Basic structure of tools:

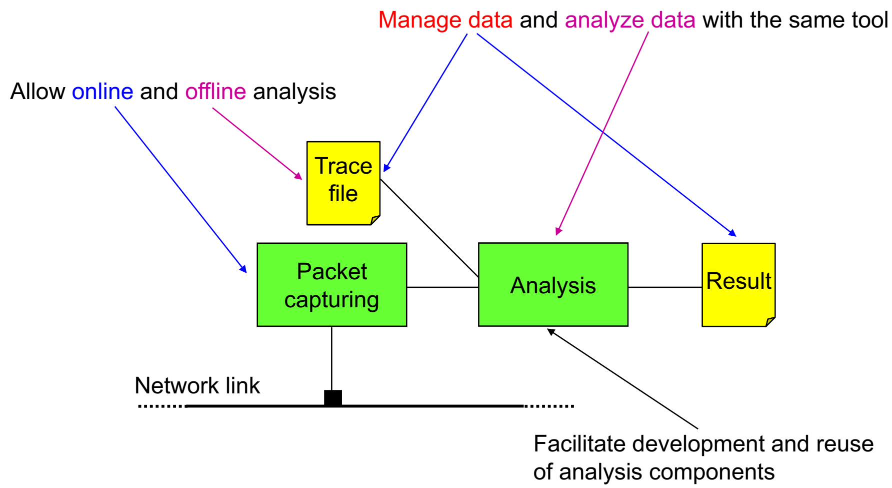


Traffic Analysis (cont.)

- Performing traffic analysis to gain new knowledge is an iterative process:



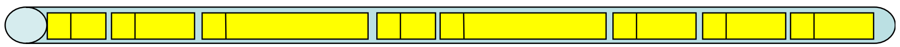
Expectations (cont.)



Expectations (cont.)

- Provide sufficient performance:
  - idealized gigabit/s link

    - all packets 1500 byte, TCP/IP header 64 byte
    - 42 megabit/s of header information
  - more realistic: compression of 9:1 or less



    - approx. 880 megabit/s on gigabit/s link
    - approx. 11 megabit/s for 100 megabit/s network

Approach

- Public domain DSMSs we have tested:
  - TelegraphCQ
  - STREAM
  - Borealis
  - …
- Tested systems:
  - install system
  - connect it to wrappers, i.e., sources
  - model TCP traces/streams
  - develop queries for simple but typical tasks
  - try to re-implement an existing complex tool
  - identify performance bounds

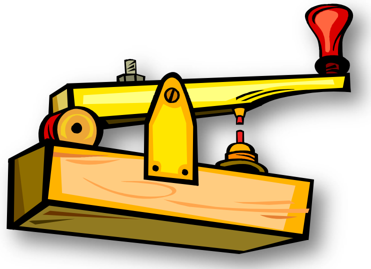
Data Stream Management Systems: Concepts from three public domain systems

INF5090, Spring 2008

Jarle Søberg

Overview

- System descriptions and query execution on:
  - TelegraphCQ
  - STREAM
  - Borealis



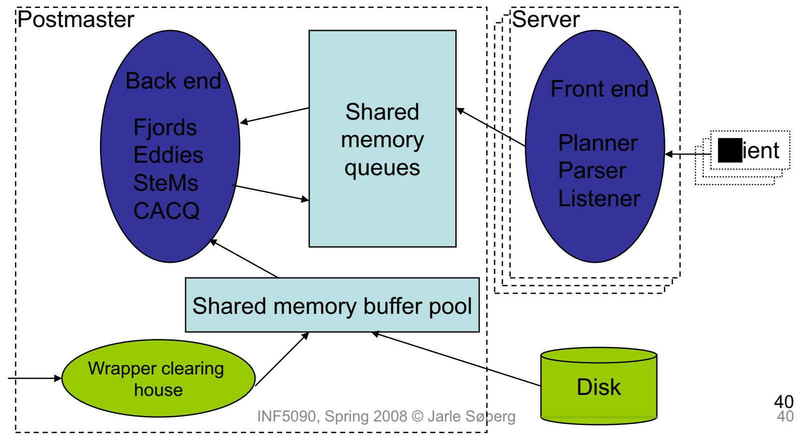
38

TelegraphCQ: Introduction

- Developed at Berkeley
- Written in C
  - Open source GNU license
- Based on the PostgreSQL DBMS
- Current version: 2.1 on PostgreSQL 7.3.2 code base
- Project closed down Summer 2006
  - Still, many interesting and important features to discuss
  - The mailing lists are still active
- The spin-off product is Truviso (.com)

39

TelegraphCQ: Overview



40

TelegraphCQ: Overview

- Based on modules
  - Query processing
  - Adaptive routing
  - Ingress and caching
- Communicate via Fjords
  - Push and pull data in pipeline fashion
  - Reduce overhead by non-blocking behavior

41

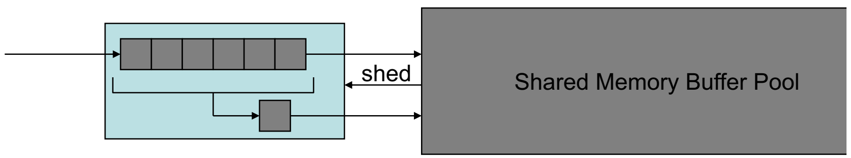
Wrappers

- Transform data to Datum items, which are understood be TelegraphCQ
- Push or pull
- Several formats
  - Comma separated format (CSV) is used by TelegraphCQ
- Contacted via TCP
- Wrapper clearing house (WCH)
  - Many connections possible
- Store streams to database if needed

42

Wrappers

- Shedded tuples, Data Triage
  - Support for dropping tuples
  - Periodically summarize tuple information



  - Runs “shadow” queries on shedded tuples
    - The queries run in parallel with the real queries

43

Windows

- Sliding

- Jumping

- Tumbling

Continuous Queries in TelegraphCQ

- Windowing supports *sliding*, *hopping*, and *jumping* behavior
  - Aggregations are important for correct results
  - Output does not start until window is reached when aggregations are used

```
SELECT stream.color, COUNT(*)
FROM stream [RANGE BY ‘9’ SLIDE BY ‘1’]
GROUP BY stream.color
```

45

Sub-queries

- Solved by using the WITH clause
  - Does not allow N depth, only 2
- Efficient sub-query execution can be done by distributing the data to several machines, using a load balancing feature called Flux
  - Operates on own ports that communicate in parallel with the data stream

```
WITH
   S AS
   ( SELECT * FROM
   ISPOutStream )

   R AS
   ( SELECT * FROM
   ISPInStream )

(SELECT * FROM S, R WHERE
   S.dest = R.dest
);
```


46

Introspective queries

- It is possible to run queries that investigate what happens inside the system
  - tcq_queries
  - tcq_operators
  - tcq_queues
- The results are also data streams that can be queried and used together with other streams
  - E.g. Get a stream of system memory usage and combine these with the length of the queues in TelegraphCQ

47

OR operator

- Not understood
- Complicates the query writing
  - “Give me all the sensor readings that have a temperature less than 0 degrees or above 100 degrees. Nothing else is interesting.”
  - Need to find clever ways of going around this issue when writing the queries

48

Data stream types

- The literature presents at least three different types of streams:
  - Insert streams
    - Only new elements are displayed
  - Relation streams
    - All elements in the window are displayed (includes the new elements)
  - Delete streams
    - Only deleted elements are displayed (those that are pushed out of the window)

Data stream types

- TelegraphCQ only offers relation streams
  - When a tuple is lost due to time constraints, it is overwritten
  - An example query that might be interesting is: "Give me all the tuples that are not joined with any other tuples due to timeouts."
    - We have two streams and match e.g. source and destination tuples
    - We want to extract the matching tuples and see which tuples that do not match

Overview

- System descriptions and query execution on:
  - TelegraphCQ
  - STREAM
    - (Based on a presentation made by Kjetil Hernes)
  - Borealis

Stanford Stream Data Manager

- Developed by the STREAM group at Stanford University
- Developed as a general purpose DSMS
- Written in C++
- Ended in 2006
- No known spin offs

Abstract Semantics

- Two data types:
  - Streams
  - Relations
- Windows are only sliding
  - Can be either time or tuple based
- Three classes of operators:
  - Stream-to-relation
    - Takes a stream as input and produces a relation as output
  - Relation-to-relation
    - Takes one or more relations as input and produces a relation as output
  - Relation-to-stream
    - Takes a relation as input and produces a stream as output
    - Insert, relation, or delete

53

Continuous Query Language

- STREAM's continuous query language is named CQL
- CQL consists of operators from the three classes

SELECT * FROM S1 [ROWS 100], S2 [RANGE 1 MINUTE]
WHERE S1.A = S2.A AND S1.A < 50

54

Operators: Relation-to-Relation

- CQL uses SQL constructs to express its relation-to-relation operators
- The current CQL implementations only offers a small subset of the SQL operators that are usually provided through a DBMS
- Must be written in beforehand and run
  - Only one process is running at all time
  - Removes dynamicity
- A load shedder does not exist, even though presented in the literature
  - Uses a sample() operator instead

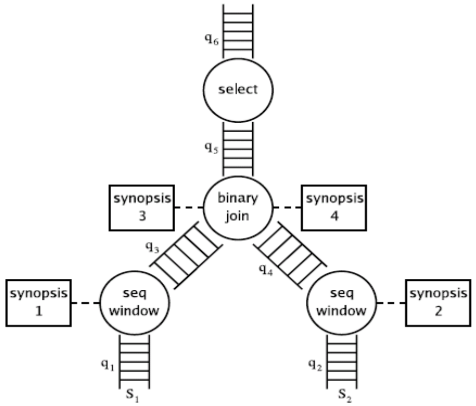
55

Query Plan

- A query plan has three types of components:
  - Operators
  - Queues
  - Synopses
    - Temporary storage
- When a query plan is executed, a *scheduler* selects operators in the query plan to execute in turn

56

Query Plan Example



57

Performance Issues

- Resource sharing
  - Queues
  - Synopsis
  - This can be distributed over several machines
- Exploiting Constraints
- Operator Scheduling
- Adaptivity
- Approximation

58

Example STREAM query

Top-k Traffic Query: Monitor the source-destination pairs in the top 5 percentile in terms of total traffic in the past 20 minutes over a backbone link B.

```
Load:
     Select   srcIP, destIP, Sum(len) as traffic
    From     Packets [Range 20 Minute]
    Where    colID = 'B'
    Group By srcIP, destIP

Q:
    Select   srcIP, destIP, traffic
    From     Load as L1
    Where    (Select Count(*)
          From   Load as L2
          Where  L2.traffic < L1.traffic) >
         (Select 0.95 * Count(*)
          From Load)
    Order By traffic
```

59

Overview

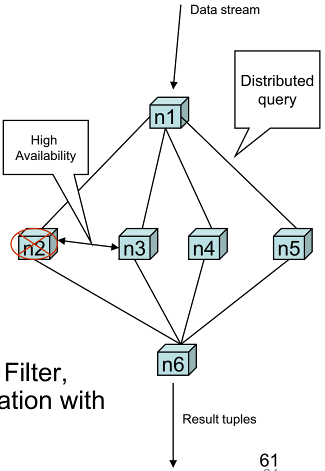
- System descriptions and query execution on:
  - TelegraphCQ
  - STREAM
  - Borealis

http://nn.wikipedia.org/wiki/Fil:Aurora_borealis_in_Alaska.jpg

60

Borealis

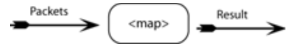
- Stream processing engine (SPE)
  - Academic research / Public domain
  - Distributed queries
  - General purpose
    - Multi-player first person shooter game
    - Network monitoring
- Continuous query language
  - Operator boxes and stream arrows
  - XML + GUI
  - E.g., operators: Map, Aggregate, Join, Filter, Random Drop and operators for integration with statically stored tables



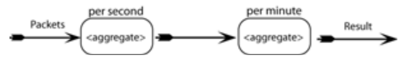
61

Design

- Simple mapping:



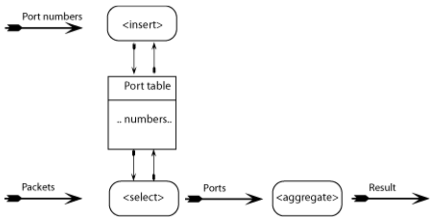
- Average load and packet count



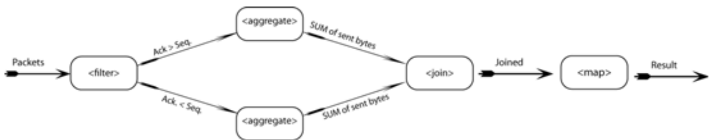
62

Cont. Design

– Port destination cont



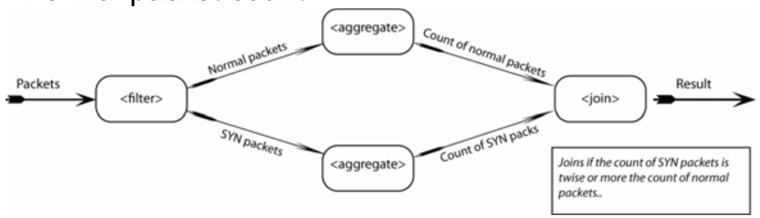
– Exchanged bytes over connections



63

Cont. Design

- SYN Flood attack (Several hosts initiate half-open connections to a server so that it has to deny service to others)
- Identifies the relation between the count of SYN packets and normal packets (Non-SYN). Joins aggregated tuples if SYN count is twice or more the normal packet count.



64

Cont. Design

```
<box name="synfilter"   type="filter" >
    <in   stream="Packet"    />
    <out  stream="Syn" />
    <out  stream="Normal" />
    <parameter    name="expression.0”     value="syn == 1"/>
    <parameter    name="pass-on-false-port”                    value="1" />
  </box>

<box name="Normalcount"   type="aggregate" >
    <in   stream="Normal" />
    <out  stream="Aggregatenormal" />
    <parameter   name="aggregate-function.0”                   value="count()"  />
    <parameter   name="aggregate-function-output-name.0”       value="count"  />
    <parameter   name="window-size-by”   value="VALUES"   />
    <parameter   name="window-size”      value="1"       />
    <parameter   name="advance”          value="1"       />
    <parameter   name="order-by”         value="FIELD"    />
    <parameter   name="order-on-field"   value="timestamp" />
  </box>

  <box name="Syncount"   type="aggregate" >
    <in   stream="Syn" />
    <out  stream="Aggregatesyn" />
    <parameter   name="aggregate-function.0”                   value="count()"  />
    <parameter   name="aggregate-function-output-name.0”       value="count"  />
    <parameter   name="window-size-by”   value="VALUES"   />
    <parameter   name="window-size”      value="1"       />
    <parameter   name="advance”          value="1"       />
    <parameter   name="order-by”         value="FIELD"    />
    <parameter   name="order-on-field”   value="timestamp" />
  </box>
```

```
<box name="SynfloodJoin" type="join" >
    <in  stream="AggregateNormal" />
    <in  stream="AggregateSyn" />
    <out stream="Result" />

    <parameter   name="predicate"           value = "left.count * 2 &lt; right.count
                                            and left.count &gt; 0" />
    <parameter   name="left-buffer-size"    value = "1" />
    <parameter   name="left-order-by"       value = "VALUES" />
    <parameter   name="left-order-on-field”  value = "timestamp" />
    <parameter   name="right-buffer-size”   value = "1" />
    <parameter   name="right-order-by”      value = "VALUES" />
    <parameter   name="right-order-on-field”  value = "timestamp" />
    <parameter   name="out-field-name.0” value="timestamp" />
    <parameter   name="out-field.0"          value="left.timestamp" />
    <parameter   name="out-field-name.1"  value="ratio" />
    <parameter   name="out-field.1”          value="right.count / left.count" />
    <parameter   name="out-field-name.2" value="syn" />
    <parameter   name="out-field.2"          value="right.count" />
    <parameter   name="out-field-name.3” value="normal" />
    <parameter   name="out-field.3"          value="left.count" />
 </box>
```


65

DSMS Benchmarking

Morten Lindeberg
University of Oslo

Agenda

- Introduction
- DSMS Recap
- General Requirements
- Metrics
- Example: Linear Road
- Example: StreamBench

Introduction

- Benchmarking is performed in most domains where several products exist
- Motivation:
  - Identify the factor in which a DSMS can outperform a DBMS, or simply “roll-your-own scripts” in a streaming environment
  - Identify whether or not you can deploy a DSMS performing real-time analysis of a stream with tens of thousand tuples/sec
  - Compare existing DSMSs, choose the DSMS that is most fit for your task

DSMS Recap #1

- Data source:
  - Data stream
  - Possibly unbound
  - Relational tuples with attributes
- Data processing:
  - Performed in main memory
  - Except for historical queries, where streaming tuples are joined with tuples in statically stored relational tables
- Query interface:
  - E.g., extended SQL

DSMS Recap #2

- Uncontrollable arrival rate:
  - Load Shedding
  - Sampling
  - Aggregations (windowed)
  - Data reduction techniques e.g., sketching and histograms

  approximations
- Blocking operator problem:
  - Applies to joins and aggregations
  - Solutions: Windowing techniques, e.g., sliding windows.

General DSMS Requirements

1. Keep the data moving
2. Query interface e.g., extended SQL
3. Handle imperfections
4. Generate predictable outcomes
5. Integrate stored and streaming data
6. Guarantee data safety and availability
7. Partition and scale applications automatically
8. Process and respond instantaneously

[Stonebraker et al. "The 8 Requirements of Real-Time Stream Processing"]

Benchmark should ideally identify how the DSMS face these requirements!

Metrics

- Response time
  - "How long does it take for the system to produce output tuples?"
  - Challenge: Windowing!
- Accuracy
  - "How accurate the system is for a given load of data arrival and queries?"
  - Especially applies to an overloaded system, where approximations rather than correct answers are presented
  - Challenge: Need to know the exact expected result
- Scalability
  - "How much resources are needed by the system to process a given load with a defined response time and accuracy?"
  - Consumption of memory
  - Utilization of CPU

[Chaudhry et al. "Stream Data Management"]

- Throughput
  - Tuples per second
  - Relative throughput $RT_{x\%}$ (StreamBench)
- Additionally identify how and with what ease queries can be expressed

Linear Road Benchmark #1

- Master's thesis at M.I.T
- Linear City
  - Traffic in this city, is the actual workload
  - Fixed city with roads and generated traffic
  - Generated before runtime, and stored in a flat file
- Perform variable tolling
  - Based on real-time traffic and traffic congestion
  - Every vehicle is transmitting their location periodically

Linear Road Benchmark #2

- Involves both historical queries, and real-time queries
- Solves the task of a very specific problem: variable tolling
- Metric: L-factor
  - The number of highways the DSMS can support execution on, within a certain permitted time frame

Linear Road Benchmark #3

- Benchmarked Systems:
  - STREAM
  - Unknown Relational Database (Commercially available)
  - Aurora
  - IBM Stream Processing Core (SPC)
  - SCSQL (Uppsala University)

Linear Road Evaluation

- Has proven that a DSMS might outperform a commercial available database by a factor of 5*1(+)
- Only a single metric (-)
- Jim Gray states the need of "domain specific benchmarks". Variable traffic tolling might fail in most DSMS application domains (-)
- A lot of work is needed for deploying Linear Road on a DSMS benchmark, since the target of the benchmark (Linear Road) is a fairly complex application itself (-)

*1 [Arasu et al. "Linear Road: A Stream Data Management Benchmark]

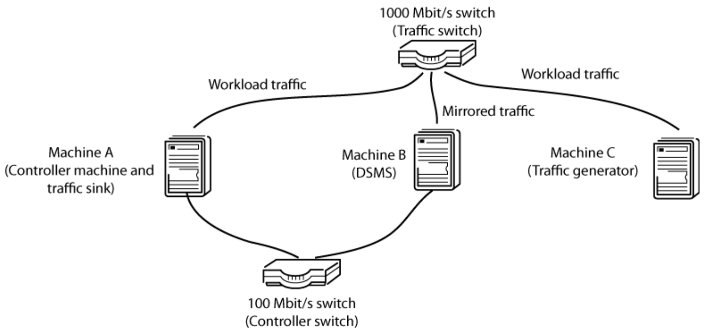
StreamBench

A Benchmark for Data Stream Management Systems used for Network Monitoring

StreamBench Motivation

- Domain specific benchmark
  - Real-time passive network monitoring
  - Traffic traces are collected and analyzed on the fly
  - TCP and IP packet headers are collected from a network interface card (NIC)
- Based upon work from three Master's theses at DMMS group, here at ifi.

StreamBench Architecture #1



StreamBench Architecture #2

- Machine B modules:
  - DSMS of subject
  - fyaf
    - Filters traffic between A and C, and sends to DSMS in CSV format
  - stim
    - Investigates the time it takes from a tuple is received at the NIC (network interface card), to a result tuple is presented by the DSMS
  - Various system monitors (e.g., top & sar)
    - Monitors the consumption of resources such as CPU and memory
- Machine A and C modules:
  - TG 2.0
    - Used for traffic generating
  - BenchmarkRunner
    - Controls the TG instances and generates traffic. Also determines workload and relative throughput

StreamBench Metrics

- Relative throughput
  - By using fyaf
  - Identification of RT $_{X\%}$, where x is the minimum percentage of successfully received tuples (network packets). The x values 100%, 98% and 95% are default
- Response time
  - By using stim
  - Need to know the DSMS behavior regarding windowing
- Accuracy
  - By looking at the DSMS output
  - Need to know the result for exact calculation
- Memory and CPU (scalability)
  - By using the Linux utilities top and sar
  - Measures of memory and CPU continuously logged during task execution

StreamBench fyaf Module

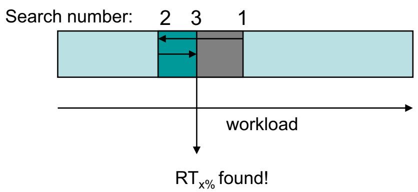
- fyaf - fyaf yet another filter
- Written in C
- Reads from NIC through the use of PCAP-library
- Filters out unwanted traffic through PCAP filter capabilities
- Converts data from PCAP into comma separated values (CSV) in strings
- Creates a TCP socket to the DSMS, used for sending the tuples
- Uses PCAP functionality to identify the number of lost tuples that the DSMS did not manage to retrieve (due to overload)

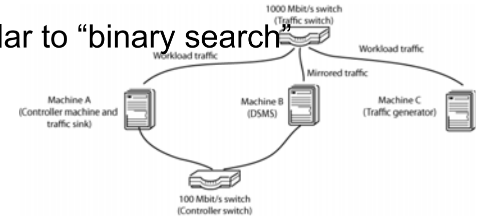
StreamBench stim Module

- stim - stim time investigation module
- Written in C
- Used to identify response time
- 3 stages:
  1. Initialization
  2. Wait for available tuples (packets) on NIC (timer is started)
  3. Wait for output on DSMS output file (timer is stopped)
- Handles windowing by “sleeping” during window is filled
- Output: the response time of the DSMS

StreamBench BenchmarkRunner Module

- Collection of Perl scripts run on multiple machines
- Controls the execution of TG 2.0, fyaf, stim, top, sar, and as well the DSMS subject of the benchmark
- Dynamically sets the workload to identify the maximum workload the DSMS can handle ($RT_{100\%}$, $RT_{98\%}$ and $RT_{95\%}$)
- Uses an approach similar to “binary search”





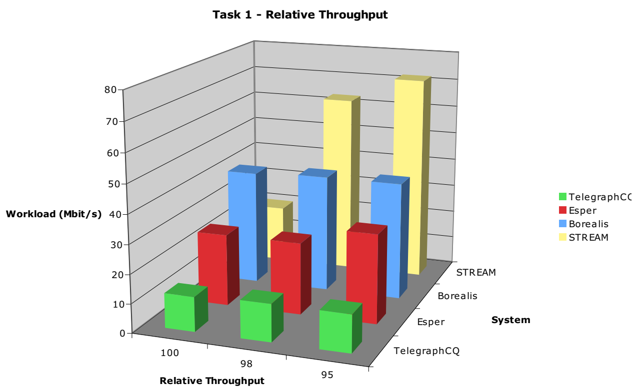
StreamBench Tasks

1. Projection of the TCP/IP header fields
   – Easy to measure response time
2. Average packet count and network load per second over a one minute interval
   – Easy to measure accuracy
3. Packet count to certain ports during the last five minutes
   – Join a stream and a static table
4. SYN flooding attacks
   – A practical task for usability
   – We investigate a simple SYN vs. non-SYN packets relation

StreamBench Results

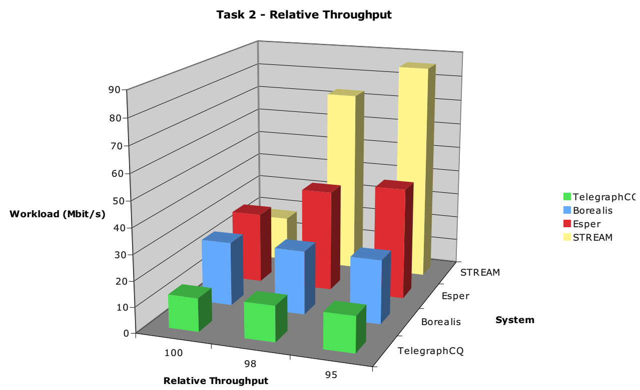
- Benchmarking of four systems:
  – TelegraphCQ - Public domain discontinued from Berkeley
  – STREAM - Public domain discontinued from Standford
  – Borealis - Public domain from Brandeis, Brown and M.I.T.
  – Esper - Open Source Commercial Java library from EsperTech
- Esper is only partly benchmarked. Further testing is a possible master's thesis topic!
- Another master's thesis topic is to benchmark SCSQL from Uppsala University

StreamBench Results Task 1



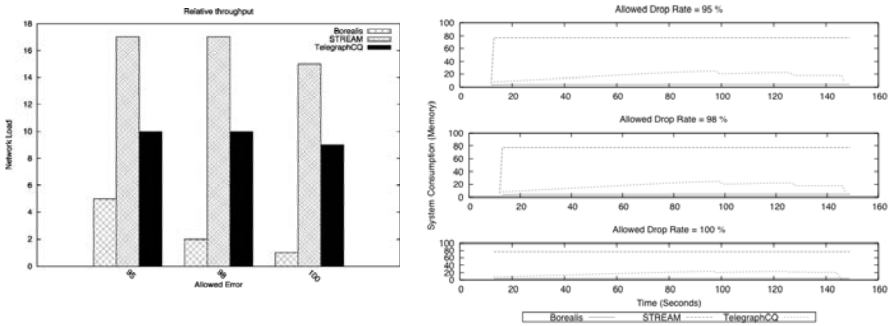
*select * from Packets;*

StreamBench Results Task 2



select count(*), avg(totallength) from Packets;

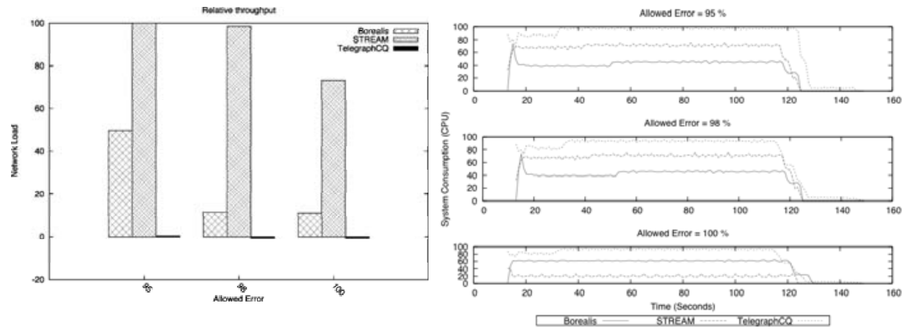
StreamBench Results Task 3



select count(*) from Packets, Ports

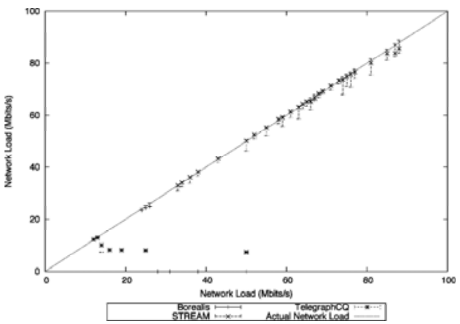
where Packets.destport = Ports.nr group by destport

StreamBench Results Task 4



StreamBench Results Accuracy

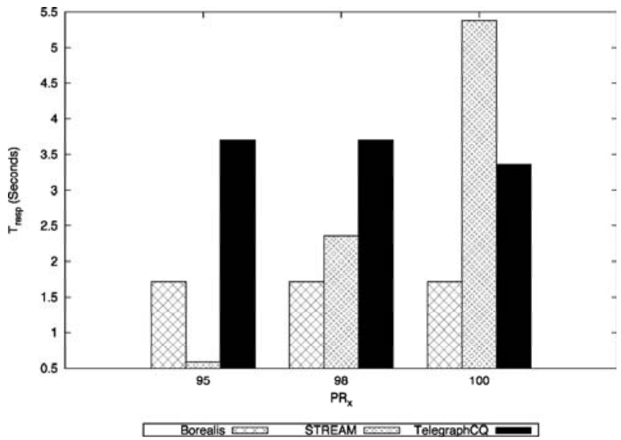
- Applies for Task 2
- We see the accuracy of the results at increasing workload / network traffic
- TelegraphCQ delivers very inaccurate results when overloaded



StreamBench Results Response Time

Applies for Task1 (projection)

PR = RT



We're done. Thank you!