Design, Implementation, and Evaluation of Network Monitoring Tasks with the TelegraphCQ Data Stream Management System

INF5100, Autumn 2006
Jarle Søberg
Content

- Introduction
- Streaming Applications
- Data Stream Management Systems
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
Introduction

- Network monitoring is important
  - On-line
  - Real-time
  - A lot of programs exist
    - ZoneRanger
    - SiteAlive
    - InMon
    - Netflow
    - MeasureNet
    - Netdisco
    - Internet Detective
    - and so on …

Source: http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html
Introduction

- Different solutions based on the protocol standards
  - Cumbersome?
  - Too specific?
  - How about new protocols?
  - Add-ons?
  - Proprietary solutions?
  - Written badly?
Introduction

- Declarative languages
  - Database management systems (SQL)
  - The web (HTML)
  - Prolog
  - What is solved?

- Use declarative languages to perform network monitoring!
Introduction

- Data stream management systems (DSMSs)
  - E.g. SQL syntax
  - Easy to learn
  - Don’t have to be a programming expert
  - Several DSMSs available (look at earlier INF5100 lecture)
    - STREAM
    - Gigascope
    - Borealis ⊆ Aurora, Medusa
    - TelegraphCQ
Introduction

- We have chosen to evaluate the performance of TelegraphCQ in an on-line network monitoring scenario
  - Build a general understanding/give a reminder of
    - Network monitoring
    - DSMSs
    - TelegraphCQ
Introduction

- Create a set of network monitoring tasks
- Create a framework for the network monitoring
- Evaluate the performance of TelegraphCQ on the set of tasks provided
- Discuss and show results
- Conclude the performance evaluation
Content

- Introduction
- **Streaming Applications**
- Data Stream Management Systems
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
Streaming Applications Revisited

- The *data stream model*
  - A contrast to the traditional stored data records
    - E.g. in databases
  - On-line
  - Un-ordered
  - Un-bounded in size
  - Append-only
Streaming Applications Overview

- Streaming Applications
  - Pull-Based
  - Sensor Networks
  - Push-Based
    - Network Monitoring
      - Network Traffic Measurement
        - Active
        - Passive
      - Others
    - Network Traffic Analysis
Pull-Based Applications

- Sensor networks revisited
  - Pull data from environment at certain intervals
  - Wireless
  - Minimize power usage
  - Aggregate data before sending
  - Locate and communicate with other sensors
  - Send data streams to access points where the data is further analyzed
Push-Based Applications

- Network monitoring
  - Data packets
    - Protocol headers
      - Header fields
Content

- Introduction
- Streaming Applications
- **Data Stream Management Systems**
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
Data Stream Management Systems (DSMSs) Revisited

- Introduction
- Database management systems (DBMSs) versus DSMSs
  - A comparison
- Issues in DSMS
  - Continuous queries and windows
  - Approximation and optimization
  - Query languages
- Examples of DSMSs
DSMSs: Introduction

- Pose queries on a stream of data
- Extract information in real-time
- Compared to DBMSs for simplicity
  - A data packet can be viewed as a tuple
  - The header(s’) fields can be viewed as attributes
  - Still, there are some critical differences between the two system types
DBMSs  versus  DSMSs

- Persistent storage
- Transient queries
- Random access and finite data sets
- Unbounded storage
- Correct answers
- Optimize once

- Transient streams
- Persistent queries
- Sequential access and possibly unlimited data streams
- Memory limitations
- Throw away tuples and aggregate at high loads
- Adaptive

INF5100, Autumn 2006 © Jarle Søberg
Transient and persistent

- Continuous queries
  - Run query at each instance of time
    - Investigate each tuple
  - Monotonic and append-only due to sequential access

*Give me all the messages that have not been replied to*

- Blocking of the data stream!!
  - Must be avoided
  - Stream is stopped and tuples are shedded
Introducing

Give me all messages that have not been given a reply within five minutes after it has been sent

- Partition data stream into smaller parts
- Use *windows* to unblock
- Different types:
  - Sliding
  - Jumping
  - Tumbling
Windows

- Sliding
- Jumping
- Tumbling
Storage?

- Disk I/O is expensive!
  - Can not store tuples to disk
  - Can only watch tuples once

- Dependent on memory and window size
- Query processing must be simple

- Windows solve the blocking problem
Content

- Introduction
- Streaming Applications
- Data Stream Management Systems
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
TelegraphCQ

- Introduction and overview
- Description of concepts
  - Wrappers
  - Fjords
  - Eddies
  - SteMs
  - CACQ
  - Other features
- A practical overview
- Limitations
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
  - Each group has a running copy on dmms-lab107
- Closed down Summer 2006
  - Still, many interesting and important features to discuss
TelegraphCQ: Overview

Shared memory buffer pool

Wrapper clearing house

Postmaster

Back end
Fjords
Eddies
SteMs
CACQ

Server

Front end
Planner
Parser
Listener

Disk

Client
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
Wrappers

- Transform data to Datum items
- 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
Wrappers

- Shedded tuples, Data Triage
  - Support for dropping tuples
    - Look at Vera’s presentation about methods
  - Periodically summarize tuple information
  - Runs “shadow” queries on shedded tuples
    - Described later
Eddies

- DBMSs
  - Query plan created once
  - E.g. \( R'' S'' T \) joined on some attributes may give this plan:

  - Ok, as long as data set is finite and pulled
Eddies

- How about pushed data?

Blocking or throwing away tuples is unavoidable!
Eddies

- A reconfiguration is necessary

  - Might be much changes in the different streams
  - Reconfiguration may take long time
  - Not dynamic enough
An alternative is to use an eddy:

- Dynamic on a tuple-per-tuple basis
- Adaptive to changes in the stream
Eddies: Details

- Bitmap per tuple represents each operator
  - ready and done bits
  - The ready bits specifies the operators the tuple should visit
  - Tuple is ready for output when all done bits are set
  - Manipulate bits to set a route for a tuple
  - On creation of new tuples due to e.g. joins: OR the bitmaps
Eddies: Routing policy

- Priority scheme
  - Tuples coming from an operator = high priority
    - Prevents starvation
- Originally: Back-pressure
  - Self regulating due to queuing
  - Naïve, hence not optimal
- Extended to lottery scheduling
Eddies: Lottery scheduling

- Each operator has ticket account
  - Credited for each arriving tuple
  - Debited for each leaving tuple
- Lottery among available operators
  - Empty in-queue: Fast operators
  - High number of tickets: Low selectivity operators
Eddies: Lottery scheduling

- Low selectivity operators
  - Win even if the operator is slowing down
  - Expand with a window scheme
    - Banked tickets
    - Escrow tickets

window
Eddies

- Works for single query environments
- Simple and adaptive
- May still not be optimal with respect to dynamic changes over e.g. a single join
- Extend the eddy’s strength by introducing state modules (SteMs)
SteMs

- Split joins in two
  - Dynamic
  - Send build tuples
    - Build hash tables
  - Send probe tuples
    - Look for matches
SteMs

- Any possible problems?
  - Two equal intermediate tuples!

- Solved by globally unique sequence number
  - Only youngest tuples allowed to match
SteMs: Issues

- SteMs are implemented using hash tables
  - Only equi-joins work properly

- Alternatively, use B-trees
  - Can correctly express more: “<>”, “>>”, “<=”, …
  - Not fast enough for conference show-off?
Eddies and SteMs

- Still single-query environment
- DSMSs aim to support many concurrent queries
- This feature needs to be adaptive and manage creation and deletion of queries in real-time
- Optimization is proven NP-hard
Introducing CACQ

- Continuously adaptive continuous queries
- Heuristics
  - Adding more information to the tuples
  - Creating even more meta information
- Avoid sending same singleton and intermediate tuples to same operators
- First of all: Use grouped filters!
CACQ: Grouped Filters

- Module for early filtering of selection predicates
  - For example:
    ```sql
    SELECT *
    FROM stream
    WHERE stream.a = 7
    ```
  - All tuples without `stream.a = 7` are not sent to the eddy
  - Includes “>”, “<”, and “≠”, as well
The CACQ Tuple

- Extended the eddy tuple to include bitmaps for `queriesCompleted` and `sourceId`
- The `queriesCompleted` bitmap
  - Represents the queries
  - Shows a lineage of the tuple
- The `sourceId` bitmap
  - Source when queries do not share tuples
Eddies, SteMs, and CACQ: Issues

- Bitmap statically configured
  - Faster, but not dynamic

- Much overhead experienced by the developers
  - Tuple-by-tuple processing takes time
  - *Batching tuples* are suggested
    - Static for shorter periods
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
```

START OUTPUT!
Other Information

● Pros:
  - Introspective streams
  - Sub-queries, to some extent
  - Shadow queries for Data Triage tuples

● Cons:
  - OR is not understood
  - Only istreams, and not dstreams
  - Only six ANDs between SteMs
  - TelegraphCQ is very unstable at high pressure
Content

- Introduction
- Streaming Applications
- Data Stream Management Systems
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
Query Design and Implementation

- The IP/TCP header definition
- Three tasks for evaluation
  - Two simple
  - One complex and large
  - Show how we think when creating these tasks
The IP/TCP Stream

- All TCP/IP header fields are included
- Use TelegraphCQ’s data types
  - Try to save space
  - Change most fields into integer values
    - smallint (16), int (32), and bigint (64) (all signed)
  - Use the cidr data type for IP addresses
  - Option fields as text
  - Control bits as char(1)
- Eddy and CACQ bits are added as well
Task 1

- Measure the average load of packets and network load per second over a one minute interval
  - Two results simultaneously
  - Create two different queries
    - Sub-query
    - Non-sub-query
WITH

streams.task1 AS
(
  SELECT
    COUNT(*), SUM(totalLength), wtime(*)
  FROM
    streams.iptcp
    [RANGE BY '1 second' SLIDE BY '1 second']
)

(SELECT
  AVG(totalNum), AVG(totalLength)*8
FROM
  streams.task1
  [RANGE BY '1 minute' SLIDE BY '1 second']);
Task 1: Non-sub-query (task_1.2)

```
SELECT COUNT(*)/60, AVG(s.totalLength)
FROM streams.iptcp AS s
[RANGE BY '1 minute' SLIDE BY '1 second'];
```
Task 2

- How many packets have been sent to certain ports during the last five minutes?
  - Join between table and stream
  - Which ports are interesting?
Task 2’s query

\begin{verbatim}
SELECT wtime(*), streams.iptcp.destPort, COUNT(*)
FROM streams.iptcp
[RANGE BY '5 minutes' SLIDE BY '1 second'],
ports
WHERE ports.port = streams.iptcp.destPort
GROUP BY streams.iptcp.destPort;
\end{verbatim}
Task 3: A Theoretical Approach

- How many bytes have been exchanged on each connection during the last ten seconds?
  - Identify connections
    `<sourceIP, sourcePort, destIP, destPort>`
    - We try to complicate the connection identification by looking at connection establishment
  - Investigate payload

- Overall task: Investigate protocol requirement
Task 3: Connection establishment

- The 3-way handshake

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>3</td>
</tr>
<tr>
<td>ACK</td>
<td>10</td>
</tr>
<tr>
<td>4301</td>
<td>2411</td>
</tr>
</tbody>
</table>

Sequence number
Acknowledgement number

Closed Listen SYN-sent SYN-received Established Established
Express establishment with query

- Similar conditions as for STREAM

- Time window lasting for a given period

- Merge 3-way handshake
Task 3: Challenges

- All connections in first interval are not registered
- There are time limitations for how long a connection can be stored
- No possibility of storing a connection in TelegraphCQ and remove when connection is closed
- TelegraphCQ only supports six SteMs concurrently, needing to reduce the number of conditions
System Implementation: fyaf

- Simple filter for transforming binary packets to comma separated values (CSVs) understood by TelegraphCQ
- Use the `pcap` interface
- Monitors the stream and itself
  - Harder to implement such behavior using Linux programs like `sed` and `awk`
  - More control and less possible sources of failure
- Runs two threads to stop after a time-to-live after the final tuple
  - Precise with regard to start and stop
- Batch monitor
- Higher throughput than the DSMSs
Experiment Setup

Computer A
  NIC
  TG
  Payload

Header

Computer B
  TG
  NIC
  fyaf
  DSMS
Content

- Introduction
- Streaming Applications
- Data Stream Management Systems
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
Performance evaluation

- **Metrics**
  - Relative throughput
    - How much data the DSMS receives versus how much it manages to compute
  - Accuracy
    - Are the results correct?
  - Consumption
    - Memory
    - CPU
Performance evaluation

- Factors
  - Network load
- Evaluation technique
  - Measurements
Performance evaluation

- **Workload selection**
  - TCP packets
    - The TG traffic generator
      - Manages to create accurate loads up to approx. 10 Mbits/s (according to `sar` output)
    - Packet size of 576 bytes
    - How is this for the eddy with respect to load shedding?
TelegraphCQ Monitors

- Use shadow queries for shedded tuples
  - Both count kept and dropped tuples
    - Gives relative throughput
  - Use introspective queries
TelegraphCQ Monitors

- Use introspective streams
  - Queries about inner life in TelegraphCQ
    - Queues
    - Modules
  - Not very much implemented
    - Hard to obtain any useful information
Simple Filtering Task

- Three tasks projecting “*”, “sourceip, sourceport, destip, destport”, “destport”
- Using both introspective queries and not
  - Intro: called task_101, task_101.1, and task_101.2
  - Non-intro: called task_101.3, task_101.4, and task_101.5
Relative Throughput: Simple filtering
Load Shedding Accuracy

- See how accurate TelegraphCQ is
  - fyaf should not drop any packets since TelegraphCQ supports load shedding
  - We investigated log files from both TelegraphCQ and fyaf on the filtering task
    - For both introspective (lines) and non-introspective tasks (linespoints)
  - Found relation by dividing between fyaf’s and TelegraphCQ’s results
Load Shedding Accuracy

![Graph showing Load Shedding Accuracy over Network load (Mbits/s)].

- The graph plots RT latency (RT latency) against Network load (Mbits/s).
- Key for graph lines:
  - "intro": sourceip, destip, sourceport, destport
  - "intro": destip
  - "sourceip, destip, sourceport, destport"
  - "destip"
Conclusion, Filtering

- Projecting few attributes using introspective queries seems to be most accurate
  - So we do this for the remaining tasks
Task 1

-Measure the average load of packets and network load per second over a one minute interval

-\texttt{task\_1.1} with sub-query
-\texttt{task\_1.2} without sub-query
Task 1: Relative throughput

![Graph showing relative throughput vs network load](image)
Task 1: Relative throughput

- task_1.1 probably faster because of the aggregation

Still, not that much faster
- If one eddy, the eddy still has to manage a lot of tuples
- Eddy versus SteMs
- Remember how and where the shedding is performed…
Task 1: Accuracy
Task 1

- TelegraphCQ manages to perform simple aggregations to investigate a stream
- Low relative throughput
  - Can possibly use shedded tuple information in real-time to adjust for failure
  - Can use a sub-query to aggregate smaller partitions of the stream
    - Still uncertain how large this window should be
Task 2

- **How many packets have been sent to certain ports during the last five minutes?**
- Run at three different table sizes
  - 65536, 1024, and 1
  - SteMs use hash lookups at O(1)
- **task_2.1**: 65536 ports
- **task_2.2**: 1025 ports
- **task_2.3**: 1 port
Task 2: Relative Throughput

![Relative Throughput Graph](image)
Task 2: Relative throughput, task_2.1
Task 2: Accuracy, task_2.1
Task 2: Accuracy

- Use `sar` to verify accuracy for 1 Mbits/s
- What do we see?
  - Memory and CPU?
  - Internal adaptation?
  - Some kind of other adaptation?
    - Run for one hour to investigate patterns
Task 2: Equilibrium
Task 2: Equilibrium

- At 5 Mbits/s
  - Converge to approx 188 packets each second
    - About 1 Mbits/s
  - Analytical modeling may help predicting the future behavior of the relative throughput

- Future work: Look at aggregations!
  - Though, remember throughput from other tasks…
Content

- Introduction
- Streaming Applications
- Data Stream Management Systems
- TelegraphCQ
- Query Design
- System Implementation
- Performance Evaluation
- Conclusion
Conclusions about TelegraphCQ

- Ok support for basic requirements
  - Problems expressing protocols
    - E.g. connection establishment
- Not that fast for high traffic monitoring
  - 2-2.5 Mbits/s just for filtering in our current setup
  - Has to perform heavy aggregations
    - Loose fine granularity information, e.g. single packets
- Accurate
- Unstable
Conclusion about DSMSs and network monitoring

- Interesting application for DSMSs
  - Simple way of describing data streams
  - Still interesting to look at several other DSMSs for applicability, expressiveness, and accuracy

- Ideas:
  - Declarative protocol descriptions in dynamic and self aware and self configuring networks?
Questions?