NetStore: An Efficient Storage Infrastructure For Network Forensics And Monitoring

Paul Giura and Nasir Memon
Polytechnic Institute of NYU
Did host X contacted a malicious site today, this week, this month?
How is done?

- Use Flow Data from border router
- Data stored in a row-store DBMS table

<table>
<thead>
<tr>
<th>SourceIP</th>
<th>DestIP</th>
<th>SourcePort</th>
<th>DestPort</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.238.2.1</td>
<td>11.11.14.1</td>
<td>21044</td>
<td>2397</td>
<td>6</td>
</tr>
<tr>
<td>128.238.1.2</td>
<td>22.7.3.12</td>
<td>6118</td>
<td>433</td>
<td>17</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Build indexes on few attributes
- Query data using SQL
- Using indexes -> low data insertion rate
- Loading unused data -> long query runtime
The Problem

How to store large amounts of network flow data on disk to keep high insertion rate and quickly answer monitoring and forensic queries?
### Solution: Column-Store

| SourceIP    | DestIP     | SourcePort | DestPort | Protocol | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128.238.2.1</td>
<td>11.11.14.1</td>
<td>21044</td>
<td>2397</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>128.238.1.2</td>
<td>22.7.3.12</td>
<td>6118</td>
<td>433</td>
<td>17</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Column-Store DBMS are general purpose**
Solution: Column-Store

- Column-Store DBMS are general purpose
- Auxiliary data for tuple reconstruction
- Overhead for insertion and query
Better Solution: NetStore

- **Fast Insert**
  - No complex auxiliary data structure used
  - Light preprocessing routines
  - Append only, no update or delete

- **Fast Query**
  - Column oriented, read just used data
  - No tuple reconstruction overhead
  - Processing in compressed format
  - Consider flow data semantics and usage
Expected Queries

- **Monitoring**
  - What happens in the network now or an hour ago, a day ago, in time window $T$?

- **Forensics**
  - What happened with host X yesterday, last week, last month, last year? Why? Who? How?
NetStore Architecture

**Modules**
- IDS
- Spam Filter
- Firewall
- Admin tools

**NetStore**
- Processing Engine
  - Write (once)
  - Read (many)
- Network Flows Column-Store

**Network**
- Traffic
- Data
- Flows
- Data
Network Flows Column-Store

TIME

Column 1  Column 2  Column n

Segment 1

Segment 2

Segment m

Index Node 1

Index Node 2

Index Node m
# IPs Inverted Index

<table>
<thead>
<tr>
<th>TIME</th>
<th>IP1</th>
<th>IP2</th>
<th>...</th>
<th>IP_{k-1}</th>
<th>IP_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>3</td>
<td>101</td>
<td>...</td>
<td>13</td>
<td>798</td>
</tr>
<tr>
<td>607</td>
<td>567</td>
<td>1232</td>
<td></td>
<td>22127</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Processing Engine

Result
- parse query
- identify time window
- identify columns
- get index nodes
- get relevant segments
- decompress segments
- process segments
- decode segments
- output result

Query

Query Processing

Network Flows Data
- reorder flows
- aggregate flows
- separate columns

Buffering

Columns
- create segments
- encode segments
- sort segments
- compress segments
- update indexes

Segmenting

Segments and Indexes

Network Flows Column-Store

Query Processing

Network Flows Data
- reorder flows
- aggregate flows
- separate columns

Buffering

Columns
- create segments
- encode segments
- sort segments
- compress segments
- update indexes

Segmenting

Segments and Indexes

Network Flows Column-Store
Compression Methods

- Run Length Encoding
- Dictionary Encoding
- Variable Byte Encoding
- Frame Of Reference
- Generic Compression (LZ77)
- No Compression
Compression Selection

- Per segment using data statistics
- Evaluate insertion and access score per segment pre method:

\[
\text{insert}(m) = \text{compress}(m) + \text{write}(m) \\
\text{access}(m) = \text{read}(m) + \text{decompress}(m)
\]

- Better evaluation using the probability of processing segment in compressed format:

\[
\text{access}(m) = \text{read}(m) + \text{decompress}(m) \cdot (1 - P(s))
\]
Operators

- Supports a subset of SQL commands
- SQL translated into basic operators
- Basic set of operators
  - `filter_segs`: filters segments based on the columns indexes data
  - `filter_atts`: filters attributes in relevant segments
  - `aggregate`: aggregates values at relevant positions in relevant segments (`sum, count, min, max`)
Evaluation

- **Hardware**
  - 2 Quad-Core CPU AMD 3rd gen Opteron 2.3 Ghz
  - 6 GB DDR2 RAM
  - 1TB HDD SATA-300 32MB Buffer 7200 rpm in RAID-Z

- **Software**
  - FreeBSD 7.2-RELEASE amd64
  - Java 1.6

- **One day (~62.5 Million flows)**
- 12 attributes, 8 GB of raw data
- Max insertion rate 1,300 flows/s

![Bar graph showing hourly data](hour_of_the_day.png)
Sample Queries

- Scanning (Q₁)
- Scanning victims (Q₂)
- Worms infected hosts (Q₃)
- SYN flooding (Q₄)
- Network statistics (Q₅)
Performance

- Open-source row-store: PostgreSQL
- Open-source column-store: LucidDB
- How many times NetStore is better:

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL/NetStore</td>
<td>10.98</td>
<td>7.98</td>
<td>2.21</td>
<td>15.46</td>
<td>1.67</td>
</tr>
<tr>
<td>LucidDB/NetStore</td>
<td>5.14</td>
<td>1.10</td>
<td>2.25</td>
<td>2.58</td>
<td>1.53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL/NetStore</th>
<th>LucidDB/NetStore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>93.6</td>
<td>6.04</td>
</tr>
</tbody>
</table>
Segment Size vs Compression Ratio

Segment Size

Compression Ratio

[Graph showing the relationship between segment size and compression ratio]
Segment Size vs Insertion Rate
General Queries

- **Spot queries (S)**
  - Return value associated with a single key

- **Range queries (R)**
  - Return values associated with a range of keys

- **Aggregation queries (A)**
  - Return the aggregation of the values for entire network

- **Spot aggregation queries (SA)**
  - Return the aggregation of the values for one key

- **Range aggregation queries (RA)**
  - Return the aggregation of the values for a range of keys
Segment Size vs Query Runtime

![Graph showing the relationship between segment size (in million records) and query runtime (in seconds). The x-axis represents segment size ranging from 0.2 to 2.0 million records, and the y-axis represents time in seconds ranging from 0 to 50. Five different lines represent different methods: S, SA, R, RA, and A. The graph indicates that as the segment size increases, the query runtime decreases for all methods.]
Conclusion

- Increasing demand of efficient storage systems for large amounts of flow data

- Current DBMS systems not designed for network forensics and monitoring workloads

- NetStore achieves 10+ times faster query answer using significantly less storage space
Thank you?
NetStore: An Efficient Storage Infrastructure For Network Forensics And Monitoring

Questions?