Aggregation and Degradation in JetStream: Streaming analytics in the wide area

Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman
Princeton University

报告人：申毅杰 2014年12月11日
Outline

• Motivation
• Solutions
  – Aggregation
  – Degradation
• Experiment
• Related work
• Conclusions
Motivation

• Target
  – Analyze data be continuously created across wide-area networks

• Challenges
  – Queries have real-time requirements
  – Available bandwidth is limited & change over time

• Goal
  – Optimize use of WAN links by exposing them to stream system
Limitation of Current systems

• Address latency in a single datacenter with high-bandwidth
  – E.g. Google MillWheel, Storm, Spark Streaming
  – Edge node backhaul all potential useful data to central location
    • High bandwidth demand
    • Limited use of edge nodes’ storage & computation
  – Developer should specify everything based on pessimistic assumption about bandwidth
    • Bandwidth is not used efficiently
JetStream’s Methodology

• Reducing the data being transferred
  – Aggregation: store & process data at edge
    • Data cube
  – Degradation: monitor available bandwidth & reduce data size at the expense of accuracy
    • Feedback control

• Application Scenarios
  – Log processing across the globe
  – Smart electric grids, highway
  – Networks of Video cameras
A Example Query

CDN Requests

How popular is every URL?

CDN Requests
Mechanism 1: Storage with aggregation

Every minute, compute request count by URL
Mechanism 2: Adaptive Degradation

Every minute, compute request count by URL
The Data Cube Model

• Cube
  – A multi-dimensional array, indexed by a set of *dimensions*, whose cells holds *aggregates*

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>12:00</th>
<th>12:01</th>
<th>12:02</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mysite.com/a">www.mysite.com/a</a></td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.mysite.com/b">www.mysite.com/b</a></td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.yoursite.com">www.yoursite.com</a></td>
<td>5</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td><a href="http://www.her-site.com">www.her-site.com</a></td>
<td>8</td>
<td>12</td>
<td>...</td>
</tr>
</tbody>
</table>

Aggregation can:
• Updates
• Roll-ups
• Merging cubes
• Summarizing cubes
Aggregates on Cubes

• Roll-up: Aggregate along some dimension

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>12:00</th>
<th>12:01</th>
<th>12:02</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mysite.com/a">www.mysite.com/a</a></td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.mysite.com/b">www.mysite.com/b</a></td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td><a href="http://www.yoursite.com">www.yoursite.com</a></td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.her-site.com">www.her-site.com</a></td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counts by URL</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mysite.com/a">www.mysite.com/a</a></td>
<td>8</td>
</tr>
<tr>
<td><a href="http://www.mysite.com/b">www.mysite.com/b</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="http://www.yoursite.com">www.yoursite.com</a></td>
<td>9</td>
</tr>
<tr>
<td><a href="http://www.her-site.com">www.her-site.com</a></td>
<td>20</td>
</tr>
</tbody>
</table>

Aggregate functions supported by JetStream should be deterministic & Order-independent
Cube Unify Storage & Aggregation

• Operators in traditional Stream Processing System
  – Stateful, maintaining state in itself
  – Store input tuples into durable buffer
    • Replay to restore state in face of Node failure
    • Or, re-scan all the data on every query

• Operators in JetStream
  – Query the cube each time and generate results
  – Cube are stored where it is generated
Degradation: The Big Picture

• Level of degradation auto-tuned to match bandwidth
Degradation Mechanisms

- Achieved via three components
  - Operators with multiple degradation level
  - Congestion monitor measures the available bandwidth
  - Policy specify how to adjust degradation level to meet bandwidth
Components of Degradation

• Degradation Operator
  – Associate with a set of degradation levels
    • E.g. roll-up across different time intervals (1s, 5s, 10s)
  – Characterize the levels with bandwidth usage
    • E.g. [1, 0.2, 0.1]

• Monitoring bandwidth
  – Attached to each queue in system
  – Network congestion
    • Insert periodic markers & get response
  – Storage bottleneck
    • Change queue length & measure the rate of queue growth
Components of Degradation

• Congestion response policies (inside a controller)
  – Several operators affect queue length
  – A single degradation technique is only useful up to a certain level
  – Several operators degradation should be combined to reach a limitation in bandwidth
  – Policy control priories or simultaneous degradation in multiple operators
Example: degradation in image sending

- By default, send all images at maximum fidelity from cameras to a central repository.
Degradation methods

- Coarsen a dimension
- Drop low-rank values
- Consistent sampling
- Synopsis approximation
Challenge: Mergeability of heterogeneous data

- Since degradation level will vary over time & vary across different nodes feeding into a single cube, no additional penalty is desired.

<table>
<thead>
<tr>
<th>Every 5</th>
<th>01 - 05</th>
<th>06 - 10</th>
<th>11 - 15</th>
<th>16 - 20</th>
<th>21 - 25</th>
<th>26 - 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every 6</td>
<td>01 - 06</td>
<td>07 - 12</td>
<td>13 - 18</td>
<td>19 - 24</td>
<td>25 - 30</td>
<td></td>
</tr>
<tr>
<td>Every 10</td>
<td>01 - 10</td>
<td>11 - 20</td>
<td>21 - 30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experiment Setup

• 80 nodes on VICCI testbed at three sites
  – Seattle
  – Atlanta
  – Germany

• (Send image) To a single union node in Princeton

• Degradation Policy
  – Drop data if insufficient Bandwidth
Without & with degradation
Related Works

• Single datacenter stream processing
  – Google MillWheel, Spark-Streaming, Storm
  – All rely on underlying fault tolerant storage system
  – Orthogonal to JetStream

• Wide area streaming system
  – Use redundant path for performance
  – Assume edge nodes has little computation ability
Conclusion

• Useful to embed aggregation and degradation abstraction in streaming systems
• Aggregation can be unified with storage
• Degradation semantic is workflow specific