Hypertable Goes Realtime at Baidu

Yang Dong
yangdong01@baidu.com
Sherlock__Yang(http://weibo.com/u/2624357843)
Agenda

• Motivation
• Related Work
• Model Design
• Evaluation
• Conclusion
Agenda

- Motivation
- Related Work
- Model Design
- Evaluation
- Conclusion
Motivation

Interval: 60 days

Interval: 1 second

FAILURE
What is Noah

- **Noah System**
  - Operational Data Store
    - System metrics (CPU, Memory, IO, Network)
    - Application metrics (Web, DB, Caches)
    - Baidu metrics (Usage, Revenue)
      - Easily visualize data over time
      - Supports complex aggregation, transformations, etc.
  - Component
    - Monitoring sub-system
    - Collection sub-system
    - ... ...
System Requirement

- **Storage Capacity**
  - 10TB~100TB
  - 100~1000 billion Records

- **Automatic Sharding**
  - Irregular data growth patterns

- **Heavy Writes**
  - 10000~30000 inserts/s

- **Fast Reads of Recent Data**
  - Recently written data should be available quickly

- **Table Scans**
  - e.g. The entire dataset will also be periodically scanned in order to perform time-based rollups
Problems of Existing Stack

- **MySQL**
  - Not inherently distributed
    - Difficult to scale
      - Irregular data growth patterns
      - Manually re-sharding data frequently
  - Table size limitation
  - Inflexible schema

- **Hadoop**
  - No support for random writes
  - Poor support for random reads
Hypertable Meets the Requirement

- Hypertable + Hadoop
  - Elasticity
  - High write throughput
  - High Availability and Disaster Recovery
  - Fault Isolation
  - Range Scans
Typical Applications of Hypertable

• Common Library
  — AsyncComm/Compression/HQL … in Hypertable

• Database (Query Engine)
  — Hypertable

• MapReduce (Batch Processing)
  — MapReduce + Hypertable

• OLAP (Online Analytical Processing)
  — MySQL + Hypertable
Agenda

- Motivation
- Related Work
- Model Design
- Evaluation
- Conclusion
Related Work

- Apache Hadoop Goes Realtime at Facebook
  - *Titan* (Facebook Messages)
  - *Puma* (Facebook Insights)
  - *ODS* (Facebook Internal Metrics)
Agenda

• Motivation
• Related Work
• Model Design
• Evaluation
• Conclusion
Model Design

- Storage System Architecture
- Functional Model
- Processing Model
- Data Model
Functional Model

- Transfer
- Transfer
- Transfer

Client IO Stub

Stream Computing Engine

Hypertable

HDFS

Noah Storage Cluster

- Client Random Read
- Client
- Client
- Client Scan

Real-time Flow

History Flow
Processing Model

Query Interface

InfoQuery <table> <rowkey_range_start> <rowkey_range_end> <column_family> <to_file> <human|machine> [max_lines]

- table: table name
- rowkey_range_start & end: row key string range
- column_family: monitor_data_status is "value", other tables are "value_pack"
- to_file: file by result written into
- human|machine: text or binary
- max_lines: default < 2w
Processing Model

• Query Mode
  — Normal Query
    • Hostname + MonitorItemIDs + Interval -> Results Graph
    • Hostnames + MonitorItemIDs -> Sorted Latest Results
    • Hostnames -> Status Results
  — Special Query
    • Many Hosts + MonitorItemIDs + Interval: Multiple queries
    • Long Interval: default <= 2w Recods
    • Low Latency: No guarantee
    • Batch queries continuously: Influence to system
Data Model

• Tradeoff
  — Query efficiency
  — Record volume

<table>
<thead>
<tr>
<th>Tables</th>
<th>Row Key</th>
<th>Column</th>
<th>Version</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>History Tables</td>
<td>Hostname + Timestamp</td>
<td>MonitorItem +MonitorItem*</td>
<td>1</td>
<td>0.5 ~ 24 months</td>
</tr>
<tr>
<td>Latest Table</td>
<td>Hostname + MonitorItemID</td>
<td>MonitorItem +MonitorItem*</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Status Table</td>
<td>Hostname</td>
<td>MonitorItem</td>
<td>10</td>
<td>24 months</td>
</tr>
</tbody>
</table>
Agenda

• Motivation
• Related Work
• Model Design
• Evaluation
• Conclusion
Evaluation

• High Availability
• Memory Usage
• Read/Write Performance
High Availability

• Challenge
  – Recovery
    • No support sync for append operation in Hadoop-v2 (based on 0.18.2)
  – Load Balance
    • No implementation in Hypertable-B2 (based on 0.9.2.0)
High Availability

- Improvements
  - Recovery
    - Data in HDFS, Log in LocalFS
      - Local-Recovery
    - Centralized Meta Ranges
      - 2048PB User-Data index by 16TB Meta-Data
    - Master Standby
    - Application Remove-duplicates
  - Load Balance
    - Split-driven
    - Manual online and offline operations
High Availability

• Deployment Model (before improvement)
High Availability

- Deployment Model (after improvement)

![Diagram of High Availability System]

- Hyperspace
- Meta Range Server
- User Range Server
- Master
- Master (Standby)
- Meta Table (e.g., HDFS, KFS...)
- User Table
- Distributed (e.g., HDFS, KFS...)
- Commit Log
- Local File System
Weaknesses

• Range data managed by a single range server
  — Though no data loss, can cause periods of unavailability
  — Can be mitigated by client-side cache or memcached
  — Can minimize recovery time by active standby of cluster
Memory Usage

• Challenge
  — Hypertable is memory intensive

<table>
<thead>
<tr>
<th>Function (during execution)</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypertable::CellCache::add</td>
<td>75.6%</td>
</tr>
<tr>
<td>__gnu_cxx::new_allocator::allocate</td>
<td>18.8%</td>
</tr>
<tr>
<td>Hypertable::DynamicBuffer::grow</td>
<td>4.1%</td>
</tr>
<tr>
<td>Hypertable::IOHandlerData::handle_event</td>
<td>1.0%</td>
</tr>
<tr>
<td>Hypertable::BlockCompressionCodecLzo::BlockCompressionCodecLzo</td>
<td>0.5%</td>
</tr>
</tbody>
</table>
Memory Usage

- Improvement
  - TCMalloc
  - SimplePollMalloc
## Memory Usage

<table>
<thead>
<tr>
<th>Function (during execution)</th>
<th>Mem Usage</th>
<th>Mem Usage</th>
<th>Function (during execution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CellCachePool::get_memory</td>
<td>94.3%</td>
<td>75.6%</td>
<td>Hypertable::CellCache::add</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>__gnu_cxx::new_allocator::allocate</td>
</tr>
<tr>
<td>Hypertable::DynamicBuffer::grow</td>
<td>3.8%</td>
<td>4.1%</td>
<td>Hypertable::DynamicBuffer::grow</td>
</tr>
<tr>
<td>Hypertable::BlockCompressionCodecLzo::BlockCompressionCodecLzo</td>
<td>1.1%</td>
<td>1.0%</td>
<td>Hypertable::IOHandlerData::handle_event</td>
</tr>
<tr>
<td>Hypertable::IOHandlerData::handle_event</td>
<td>0.5%</td>
<td>0.5%</td>
<td>Hypertable::BlockCompressionCodecLzo::BlockCompressionCodecLzo</td>
</tr>
</tbody>
</table>
Memory Usage

- New/Delete vs. TCMalloc vs. PollMalloc vs. PollMalloc (with map)
Memory Usage

- Opportunity
  - Compaction Efficiency
Memory Usage

• Improvement
  – Compaction Optimization Factor
  • Cell Cache Buffer Size
  • Maintenance Thread Number

RangeServer Memory-Usage Comparison (Map)
### Read Performance

**Challenge**

Noah generates lots of random read ops

<table>
<thead>
<tr>
<th>Machine</th>
<th>Sequential Write (KB/s)</th>
<th>Random Read (KB/s)</th>
<th>Random Write (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>284191</td>
<td>1932</td>
<td>79834</td>
</tr>
<tr>
<td>X2</td>
<td>165264</td>
<td>973</td>
<td>1627</td>
</tr>
<tr>
<td>X3</td>
<td>93919</td>
<td>863</td>
<td>7256</td>
</tr>
<tr>
<td>X4</td>
<td>63565</td>
<td>471</td>
<td>23149</td>
</tr>
<tr>
<td>X5</td>
<td>55598</td>
<td>424</td>
<td>18768</td>
</tr>
<tr>
<td>X6</td>
<td>60858</td>
<td>336</td>
<td>23188</td>
</tr>
</tbody>
</table>

```
./iozone -f t3 -s 10g -c -e -w -+n -i 0
```

```
./iozone -f t3 -s 10g -c -e -w -+n -i 2
```
Read Performance

• Memory Read
  — Latency
    • ms level
  — Memory usage
    • Sparse Table: value + 40Bytes(keys + map)
    • MVCC: Old record is deleted when cell cache compaction
Read Performance

• Cache Read
  — Block Cache
    • Cellstore blocks
    • Uncompressed
  — Query Cache
    • Query results
Read Performance

• Disk Read
  — CellStore
    • 5 CSs/Range, 5*216=950 RR, 950*2=1900 RR; 500 iops, ioutil 100%
  — Bloomfilter

<table>
<thead>
<tr>
<th></th>
<th>CS 10</th>
<th></th>
<th></th>
<th>CS 5</th>
<th></th>
<th></th>
<th>CS 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>1</td>
<td>16</td>
<td>32</td>
<td>1</td>
<td>16</td>
<td>32</td>
<td>1</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Queries/s</td>
<td>3.8</td>
<td>3.8</td>
<td>2.4</td>
<td>9.8</td>
<td>13.5</td>
<td>5</td>
<td>23.1</td>
<td>52.6</td>
<td>25.6</td>
</tr>
<tr>
<td>Aggregate Queries/s</td>
<td>3.8</td>
<td>61</td>
<td>77</td>
<td>9.8</td>
<td>216</td>
<td>160</td>
<td>23.1</td>
<td>1684</td>
<td>1641</td>
</tr>
<tr>
<td>Average Latency(ms)</td>
<td>260</td>
<td>260</td>
<td>420</td>
<td>102</td>
<td>74</td>
<td>200</td>
<td>43</td>
<td>19</td>
<td>39</td>
</tr>
</tbody>
</table>
Read Performance

• Disk Read
  — Compression
    • Lzo/Quicklz/Gzip/Snappy
  — SSD/SATA/SAS
    • Read Access Time/Capacity/Price per GB/Idle or Full power/MTBF
  — Concurrent W/R
    • Resource isolation
    • Manual control
Read Performance

- Hypertable vs. Hbase

Random Query Throughput
Uniform

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Queries/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 TB</td>
<td>Hypertable: 2500, HBase: 2000</td>
</tr>
<tr>
<td>0.5 TB</td>
<td>Hypertable: 3500, HBase: 2500</td>
</tr>
</tbody>
</table>

Random Query Throughput
Zipfian

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Queries/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 TB</td>
<td>Hypertable: 8000, HBase: 5000</td>
</tr>
<tr>
<td>0.5 TB</td>
<td>Hypertable: 9000, HBase: 4000</td>
</tr>
</tbody>
</table>
Write Performance

• Challenge
  — HDFS operation pressure
  — RangeServer writing pressure

• Improvement
  — Compaction frequency adjusting
  — Row key reversal
Agenda

• Motivation
• Related Work
• Model Design
• Evaluation
• Conclusion
Conclusion

- Application level
  - Table Design
  - Load-in Strategy
  - Duplicate Removing
- High Availability
  - Metadata Centralization
  - Master Standby
  - Log/Data Separation
  - Load Balance
  - System Active Standby

- Memory Usage
  - Memory Pool
  - Compaction Strategy

- Read/Write Performance
  - Mem/SSD/SAS/SATA
  - Block/Query Caching
  - Compression Strategy
  - Resource Isolation
  - Compaction Strategy
## Hypertable versus HBase

<table>
<thead>
<tr>
<th>versus</th>
<th>Hypertable</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community</td>
<td>Hypertable</td>
<td>Apache</td>
</tr>
<tr>
<td>Company</td>
<td>Hypertable Inc</td>
<td>Cloudera, HortonWorks</td>
</tr>
<tr>
<td>Implementation Language</td>
<td>Boost C++</td>
<td>Java</td>
</tr>
<tr>
<td>Memory Management</td>
<td>Explicit Memory Management</td>
<td>Garbage Collection</td>
</tr>
<tr>
<td>Cache Management</td>
<td>Dynamic cache management</td>
<td>Java heap for caching</td>
</tr>
<tr>
<td>Performance</td>
<td>High</td>
<td>Normal</td>
</tr>
<tr>
<td>Compiler Optimization</td>
<td>Easy</td>
<td>Hard</td>
</tr>
<tr>
<td>Compression</td>
<td>Direct native compression</td>
<td>JNI-based compression</td>
</tr>
</tbody>
</table>
Thanks for your Attention!
Questions?
“畅想•交流•争鸣•聚会”是百度技术沙龙的宗旨。百度技术沙龙是由百度与InfoQ中文站定期组织的线下技术交流活动。目的是让中高端技术人员有一个相对自由的思想交流和交友沟通的平台。主要分讲师分享和OpenSpace两个关键环节，每期只关注一个焦点话题。

讲师分享和现场Q&A让大家了解百度和其他知名网站技术支持的先进实践经验，OpenSpace环节是百度技术沙龙主题的升华和展开，提供一个自由交流的平台。针对当期主题，参与者人人都可以发起话题，展开讨论。

InfoQ 策划·组织·实施

关注我们：weibo.com/infoqchina