CaSSanDra: An SSD Boosted Key-Value Store

Prashanth Menon, Tilmann Rabl, Mohammad Sadoghi (*), Hans-Arno Jacobsen
Outline

• Application Performance Management
• Cassandra and SSDs
• Extending Cassandra’s Row Cache
• Implementing a Dynamic Schema Catalogue
• Conclusions
Modern Enterprise Architecture

- Many different software systems
- Complex interactions
- Stateful systems often distributed/partitioned/replicated
- Stateless systems certainly duplicated
Application Performance Management

- Lightweight agent attached to each software system instance
- Monitors system health
- Traces transactions
- Determines root causes
- Raw APM metric:

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
<th>Timestamp</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostA/AgentX/ServletB/AverageResponseTime</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>1332988833</td>
<td>15</td>
</tr>
</tbody>
</table>
Application Performance Management

- Problem: Agents have short memory and only have a local view
  - What was the average response time for requests served by servlet X between December 18-31 2011?
  - What was the average time spent in each service/database to respond to client requests?
APM Metrics Datastore

- All agents store metric data in high write-throughput datastore
- Metric data is at a fine granularity (per-action, millisecond etc)
- User now has global view of metrics

- What is the best database to store APM metrics?
Cassandra Wins APM

Read: 95%

Throughput (Operations/sec)

Number of Nodes

Cassandra HBase Voldemort VoltDB Redis MySQL

Read: 50%

Throughput (Ops/sec)

Number of Nodes

Cassandra HBase Voldemort VoltDB Redis MySQL

• APM experiments performed by Rabl et al. [1] show Cassandra performs best for APM use case
  • In memory workloads including 95%, 50% and 5% read
  • Workloads requiring disk access with 95%, 50% and 5% reads

Cassandra

- Built at Facebook by previous Dynamo engineers
  - Open sourced to Apache in 2009
- DHT with consistent hashing
  - MD5 hash of key
  - Multiple nodes handle segments of ring for load balancing
- Dynamo distribution and replication model + BigTable storage model
Cassandra and SSDs

- Improve performance by either adding nodes or improving per-node performance

- Node performance is directly dependent on the disk I/O performance of the system

- Cassandra stores two entities on disk:
  - Commit Log
  - SSTables

- Should SSDs be used to store both?

- **We evaluated each possible configuration**
Experiment Setup

- **Server specification:**
  - 2x Intel 8-core X5450, 16GB RAM, 2x 2TB RAID0 HDD, 2x 250GB Intel x520 SSD
  - Apache Cassandra 1.10

- **Used YCSB benchmark**
  - 100M rows, 50GB total raw data, ‘latest’ distribution
  - 95% read, 5% write

- **Minimum three runs per workload, fresh data on each run**

- **Broken into phases:**
  - Data load
  - Fragmentation
  - Cache warm-up
  - Workload (> 12h process)
SSD vs. HDD

- Location of log is irrelevant
- Location of data is important
  - Dramatic performance improvement of SSD over HDD
- SSD benefits from high parallelism

<table>
<thead>
<tr>
<th>Configuration</th>
<th># of clients</th>
<th># of threads/client</th>
<th>Location of Data</th>
<th>Location of Commit Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td>2</td>
<td>RAID (HDD)</td>
<td>RAID (HDD)</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>2</td>
<td>RAID (HDD)</td>
<td>SSD</td>
</tr>
<tr>
<td>C3</td>
<td>1</td>
<td>2</td>
<td>SSD</td>
<td>RAID (HDD)</td>
</tr>
<tr>
<td>C4</td>
<td>1</td>
<td>2</td>
<td>SSD</td>
<td>SSD</td>
</tr>
<tr>
<td>C5</td>
<td>4</td>
<td>16</td>
<td>RAID (HDD)</td>
<td>RAID (HDD)</td>
</tr>
<tr>
<td>C6</td>
<td>4</td>
<td>16</td>
<td>SSD</td>
<td>SSD</td>
</tr>
</tbody>
</table>
SSD vs. HDD (II)

- SSD offers more than 7x improvement to throughput on empty disk
- SSD performance degrades by half as storage device fills up
- Filling the SSD or running it near capacity is not advisable
SSD vs. HDD: Summary

- Cassandra benefits most when storing data on SSD (not the log)
  - Location of commit log not important
- SSD performance inversely proportional to fill ratio
- Storing all data on SSD is uneconomical
  - Replacing 3TB HDD with 3x 1TB SSD is 10x more costly
  - SSDs have limited lifetime (10-50K write-erase cycles), replacement more frequently
- Rabl et al. [1] show adding node is 100% costlier, with 100% throughput improvement
- **Build hybrid system to get comparable performance for marginal cost**
Cassandra: Read + Write Path

- Write path is fast:
  1. Write update into commit log
  2. Write update into Memtable

- Memtables flush to SSTables asynchronously when full
  - Never blocks writes

- Read path can be slow:
  1. Read key-value from Memtable
  2. Read key-value from each SSTable on disk
  3. Construct merged view of row from each input source

- Each read needs to do $O(# \text{ of SSTables})$ I/O
Cassandra: SSTables

- Cassandra allows blind-writes
- Row data can be fragmented over multiple SSTables over time

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>First Name</th>
<th>Last Name</th>
<th>Age</th>
<th>Department ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>99231234</td>
<td>Prashanth</td>
<td>Menon</td>
<td>25</td>
<td>MSRG</td>
</tr>
</tbody>
</table>

- Bloom filters and indexes can potentially help
- **Ultimately, multiple fragments need to be read from disk**
Cassandra: Row Cache

- Row cache buffers full merged row in memory
- Cache miss follows regular read path, constructs merged row, brings into cache
- Makes read path faster for frequently accessed data
- Problem: Row cache occupies memory
  - Takes away precious memory from rest of system
- **Extend the row cache efficiently onto SSD**
Extended Row Cache

- Extend the row cache onto SSD
  - Chained with in-memory row cache
  - LRU in-memory, overflow onto LRU SSD row cache
- Implemented as append-only cache files
  - Efficient sequential writes
  - Fast random reads
- Zero I/O for hit in first level row cache
- One random I/O on SSD for second level row cache
Evaluation: SSD Row Cache

- **Setup:**
  - 100M rows, 50GB total data, 6GB row cache

- **Results:**
  - 75% improvement in throughput
  - 75% improvement in latency
  - RAM-only cache has too little hit ratio
Dynamic Schema

- Key-value stores covet schema-less data model
  - Very flexible, good for highly varying data
  - Schemas often change, defining up front can be detrimental

### Application Format

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Timestamp</th>
<th>Value</th>
<th>Max</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostA/AgentX/AVGResponse</td>
<td>1332988833</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

### On-Disk Format

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>HostA/AgentX/AVGResponse</th>
<th>Metric Name</th>
<th>HostA/AgentX/Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>1332988833</td>
<td>Timestamp</td>
<td>1332988849</td>
</tr>
<tr>
<td>Value</td>
<td>4</td>
<td>Value</td>
<td>5</td>
</tr>
<tr>
<td>Max</td>
<td>6</td>
<td>Max</td>
<td>7</td>
</tr>
<tr>
<td>Min</td>
<td>1</td>
<td>Min</td>
<td>1</td>
</tr>
</tbody>
</table>

- Observation: many big data applications have relatively stable schemas
  - e.g., Click stream, APM, sensor data etc.
- Redundant schemas have significant overhead in I/O and space usage
Dynamic Schema (III)

- Don’t serialize redundant schema with rows
- Extract schema from data, store on SSD, serialize schema ID with data
- Allows for large number of schemas

Old Disk Format

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>HostA/AgentX/AVGResponse</th>
<th>Timestamp</th>
<th>Value</th>
<th>Max</th>
<th>Min</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/AVGResponse</td>
<td>Timestamp</td>
<td>1332988833</td>
<td>Value</td>
<td>4</td>
<td>Max</td>
</tr>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/AVGResponse</td>
<td>Timestamp</td>
<td>1332988848</td>
<td>Value</td>
<td>5</td>
<td>Max</td>
</tr>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/Failures</td>
<td>Timestamp</td>
<td>1332988849</td>
<td>All</td>
<td>4</td>
<td>Warn</td>
</tr>
</tbody>
</table>

New Disk Format

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>HostA/AgentX/AVGResponse</th>
<th>1332988833</th>
<th>S1</th>
<th>4</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/AVGResponse</td>
<td>1332988848</td>
<td>S1</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/Failures</td>
<td>1332988849</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>HostA/AgentX/AVGResponse</th>
<th>1332988833</th>
<th>S1</th>
<th>4</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/AVGResponse</td>
<td>1332988848</td>
<td>S1</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Metric Name</td>
<td>HostA/AgentX/Failures</td>
<td>1332988849</td>
<td>S2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Evaluation: Dynamic Schema

- Setup:
  - 40M rows, variable columns 5-10 (638 schemas), 6GB row cache

- Results:
  - 10% reduction in disk usage (6.8GB vs 6GB)
  - Slightly improved throughput, stable latency

- Effective SSD usage (only random reads) & reduce I/O and space usage
Conclusions

• Storing Cassandra commit logs on SSD doesn’t help

• Managing SSDs at capacity degrades its performance

• Using SSDs as a secondary row-cache dramatically improves performance

• Extracting redundant schemas onto and SSD reduces disk space usage and required I/O
Thanks!

- Questions?

- Contact:
  - Prashanth Menon (prashanth.menon@utoronto.ca)
Future Work

• What types of tables benefit most from a dynamic schema?

• Impact of compaction on read-heavy workloads
  • How can SSDs be used to improve the performance of compaction?

• How is performance when storing only SSTable indexes on SSD?