Best Practices with Kudu: An End to End Use Case Sharing in Automobile Industry

Wei Chen (wei.c.chen@intel.com)
12/7/2017
Agenda

• Brief introduction of Kudu
• Design challenges of a Kudu based cluster
• Use case: Design Kudu cluster in automobile industry
Kudu: Storage for Fast Analytics on Fast Data
HDFS, HBase, Kudu

- **HDFS**
  - Fast batch processing, fast scan
  - Immutable, append-only

- **HBase**
  - Mutable, fast random access
  - Low throughput for analytic jobs

- **Kudu**
  - High injection speed
  - Competitive analytic speed compared to HDFS
  - A balanced solution
Kudu Overview

- Raft Consensus
  - Leader, Follower
- Tablet
  - MemoryRowSet
  - DiskRowSet
- Maintenance
  - Flush
  - Compaction
- Key uniqueness
  - Min-max Check
  - Bloom Filter Lookup
Design Challenges

• Table Schema
  • Key selection, Key order, Partition plan...
• Hardware selection
  • CPU selection, Disk selection...
• System configurations
  • Virtual Memory configurations ...
• Kudu configurations
  • Maintenance threads count ...
### Table Schema – Key selection

<table>
<thead>
<tr>
<th>Schema</th>
<th>Key 0</th>
<th>Property</th>
<th>Other Cols</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Monotonically increasing</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Random</td>
<td>...</td>
</tr>
</tbody>
</table>

- Scan query with a range filter on Key 0

![Graph showing Insertion and Scan Throughput](image_url)
Table Schema – Key order

<table>
<thead>
<tr>
<th>Schema</th>
<th>Key 0</th>
<th>Key 1</th>
<th>Other Cols</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>C</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>A</td>
<td>...</td>
</tr>
</tbody>
</table>

- A: monotonically increasing
- C: NOT monotonic (random)
- Scan query with a range filter on field A

- If the **FIRST** key is monotonic, encoded keys between DiskRowSets will be monotonic.
- Monotonic keys can help reduce bloom filter lookups, benefit throughput with heavy write workload.
- Time-series application is favored by Kudu.
Table Schema – Hash buckets

• How many hash buckets should be set for each node?
  • Injection heavy workload?
    • Not too few:
      • Too few threads
      • In-sufficient resource utilization
    • Not too many:
      • Many small RPCs, high RPC overhead, many RPC rejections
      • Too many threads
      • High context switch
  • Scan heavy workload?
    • More buckets bring higher scan concurrency, result in higher throughput.
Select CPU Type

- More cores or Higher frequency?
- It depends ...

![Workload A (Key with Timestamp)](image1)

- 1.2 GHz: 497
- 2.3 GHz: 854

![Workload B (Key with Timestamp)](image2)

- 9: 854
- 18: 1311
- 36: 1400

![Workload A (Random Keys)](image3)

- 1.2 GHz: 234
- 2.3 GHz: 326

![Workload B (Random Keys)](image4)

- 9: 326
- 18: 543
- 36: 684
Disk Types

<table>
<thead>
<tr>
<th>Case</th>
<th>WAL Disk</th>
<th>Data Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HDD</td>
<td>HDD x 4</td>
</tr>
<tr>
<td>2</td>
<td>SSD</td>
<td>HDD x 4</td>
</tr>
<tr>
<td>3</td>
<td>SSD</td>
<td>SSD x 4</td>
</tr>
</tbody>
</table>

- Only one disk can be configured for WAL, WAL disk can easily become bottleneck.
- SSD as WAL disk is recommended for write heavy applications. NVMe or Optane is better.
- Bandwidth of data disks can also become the injection bottleneck, if HDD is used.
- Scan throughput of cases with HDD as data disks is ~8 to 9 times slower than SSD.
- SSDs as data disks is recommended for most applications.
Kudu configuration

- We need to keep the data flushed in time. Keep a proper number of maintenance threads can help.
- If data disks are HDDs, more mn threads results in not too much improvement.
- If data disks are SSDs, increase mn threads to a certain amount can benefit injection throughput.
- Configure Tips:
  - Not too few: Data can't be flushed in time, memory rejection will happen.
  - Not too many: Contention can happen on data disks. Context Switch increases with more active threads.
  - SSD as data disks: Set to $data\ disk\ count \times 3$
  - HDD as data disks: No more than $data\ disk\ count \times 1$

![Graph showing Average Injection Throughput (kops/s) for HDD and SSD with different mn threads.]

Software and Services
Intel Confidential
System Technologies and Optimization
Use case: Design Kudu cluster in automobile industry
System Technologies and Optimization

Design Target & Data Pipeline

• Status metrics of cars are collected at a high frequency, then transmitted to the data center and get stored.

• High injection throughput
  • 300K records per second per Kudu server, each record is ~200 bytes

• Query latest data and quick response

• Analytic jobs executed along with injection jobs
## Schema & Cluster Settings

### Table: Column Name, Type, Key or not

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Key or not</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_id</td>
<td>string</td>
<td>Key</td>
</tr>
<tr>
<td>timestamp</td>
<td>unixtime_micros</td>
<td>Key</td>
</tr>
<tr>
<td>trip_id</td>
<td>int64</td>
<td>Key</td>
</tr>
<tr>
<td>split_id</td>
<td>int32</td>
<td>Key</td>
</tr>
<tr>
<td>metric_s0</td>
<td>string</td>
<td>value</td>
</tr>
<tr>
<td>metric_d0</td>
<td>double</td>
<td>value</td>
</tr>
<tr>
<td>metric_i0</td>
<td>int32</td>
<td>value</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Platform
- **Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz**
- 2 Sockets x 18 Cores
- 256 GB DRAM, 10Gbps Ethernet

### Storage
- HDD (WAL)
- 7 x HDDs (Data)

### OS
- CentOS 6.8

### Kudu Version
- 1.3.0-cdh5.11.0
Partition Plan

<table>
<thead>
<tr>
<th></th>
<th>Schema A</th>
<th>Schema B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition Plan</td>
<td>hash (car_id, trip_id)</td>
<td>hash (car_id, trip_id)</td>
</tr>
<tr>
<td></td>
<td><strong>hash (timestamp)</strong></td>
<td><strong>range (timestamp)</strong></td>
</tr>
<tr>
<td>Writes</td>
<td>☑ writes are spread evenly among tablets</td>
<td>all writes go to latest partition</td>
</tr>
<tr>
<td>Reads</td>
<td>time-bounded scans can <strong>NOT</strong> be pruned</td>
<td>☑ time-bounded scans can be pruned</td>
</tr>
<tr>
<td>Tablet Growth</td>
<td>tablets could grow too large</td>
<td>☑ new tablets can be added for future time periods</td>
</tr>
</tbody>
</table>

- The most common query pattern of the application:
  - SELECT AVG(metric_a), MAX(metric_a), MIN(metric_a) ... FROM car_metrics WHERE timestamp BETWEEN ts_1 AND ts_2
- For the convenience of maintenance, we propose to use range partition on timestamp, as this a more natural way.
The very first performance

- Average injection throughput per node: 104 kops/s
- Very poor performance, IOPS not smooth over time.
- With only 1 mn thread, flush can't catch up with injection, memory becomes bottleneck.

- Low CPU utilization
- Heap utilization reached soft limit (60%)
Maintenance Threads Scaling

- We increase the number of maintenance threads to 6 to keep heap utilization below software limit
- Throughput Gain: ~21%
- Heap utilization relieved
- WAL is the bottleneck
WAL disk

- Change WAL disk from HDD to Intel SATA SSD
- Throughput Gained: ~32%
- IOPS is up to 500 kops/s in the beginning, but drops over time. Why?
- WAL utilization drops soon. New bottleneck?
Bloom Filter Lookups

- Vtune profiling result shows Kudu spent many cycles checking key presence.
- Bloom filter lookups become heavy as more data are flushed into DiskRowSets.
- Compaction is heavy, indicating keys are overlapped between DiskRowSets.
- The keys are not monotonic!
### Monotonic Keys

<table>
<thead>
<tr>
<th>Key No.</th>
<th>Column Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>car_id</code></td>
<td>string</td>
</tr>
<tr>
<td>1</td>
<td><code>timestamp</code></td>
<td><code>unixtime_micros</code></td>
</tr>
</tbody>
</table>

- Set `timestamp` as the first key
- Throughput Gain: ~50%
- No bloom filter lookups were observed.
- IOPS is not smooth over time. IOPS will drop as memory reached full utilization. It will bounce back once some data are flushed and memory are freed.

---

**Average IOPS: 251 kops/s**

**Heap & Memory**

---

### Schema Design

<table>
<thead>
<tr>
<th>Key No.</th>
<th>Column Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>timestamp</code></td>
<td><code>unixtime_micros</code></td>
</tr>
<tr>
<td>1</td>
<td><code>car_id</code></td>
<td>string</td>
</tr>
</tbody>
</table>
Virtual Memory Scaling

- Reserve some memory in system:
  \( \text{vm.min_free_kbytes}=10\text{GB} \)
- Throughput gain: \(~11\%\)
- Heap utilization is high. Flush not catching up.
- Data disk utilization is high.

![Average IOPS: 277 kops/s](image)

![Heap & Memory](image)

![Data Disk](image)
Data Disks

- Change data disks from HDD x 7 to Intel SATA SSD x 4
- Throughput Gain: ~24%
- Heap utilization is fine
- WAL bottleneck all the time
NVMe

- Change WAL Disk from SSD to Intel NVMe
- Throughput Gain: ~17%
- WAL utilization is low.
- Heap utilization is high, flush not catching up.
Final Tune

Average IOPS: 543 kops/s

<table>
<thead>
<tr>
<th>Scale Item</th>
<th>Default</th>
<th>Scaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Order</td>
<td>Timestamp NOT first</td>
<td>Timestamp first</td>
</tr>
<tr>
<td>WAL Disk</td>
<td>HDD x 1</td>
<td>NVMe x 1</td>
</tr>
<tr>
<td>Data Disk</td>
<td>HDD x 7</td>
<td>SATA SSD x 5</td>
</tr>
<tr>
<td>Maintenance Thread Count</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Bucket Count Per Node</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>OS Reserved Memory</td>
<td>88 MB</td>
<td>10 GB</td>
</tr>
<tr>
<td>Average Injection Throughput Per Node</td>
<td>104 kops/s</td>
<td>543 kops/s</td>
</tr>
</tbody>
</table>
Tuning History

- Average injection throughput increased up to ~5x.
- Further tuning is still possible, like encoding and compression types, column block size, flush interval, log segment size etc.
Co-executed with Analytic Jobs

- We integrated Kudu with impala to perform analytic jobs during injection.
- As CPU is bottlenecked, injection throughput will decrease as more queries are co-executed. Injection throughput dropped ~42% with 16 queries run at the same time.
- Designed target: 300 kops/s per node, able to handle 8 or more concurrent queries.
Key Takeaways

- Kudu is a balanced storage solution. It has high injection speed and competitive analytic speed.
- Tuning is a recurrent analysis job with different scalable parameters, you need to find and solve bottlenecks in both h/w & s/w.
- Optimize Hardware components
  - CPU, Disk can both become bottleneck in injection phase.
  - Use SSD or NVMe or Optane as WAL disk.
  - SSDs are recommended for data disks in most cases.
- Optimize Software Parameters
  - Use timestamp as the first key to avoid bloom filter lookups if possible.
  - Reserve some amount of memory in OS to prevent thread stuck.
  - Set proper maintenance thread count according to memory utilization.