RocksDB

Challenges of LSM-Trees in Practice

Siying Dong

Software Engineer, Database Engineering Team @ Facebook

Sep 29, 2015
What is RocksDB?
What is RocksDB?

- Key-Value persistent store
- Point / range lookup
- Optimized for flash and memory
- C++ library
- Other language bindings
- Fork of LevelDB
RocksDB As Storage Engine of Data Management Systems

mongoDB

MySQL

Yahoo Sherpa

And many more ...
RocksDB As Storage in Applications

- Facebook: many backend services
- LinkedIn’s FollowFeed
- Apache Samza
- Iron.io
- Tango Me
- And more...
Agenda

1. RocksDB Architecture
2. Challenges
RocksDB Architecture
Log Structure Merge Tree
Level-Based Compaction

Level 0

Level 1

Level 2

Level 3

Target: 100GB

Target: 10 GB

Target: 1 GB

* These Numbers are just examples
Level-Based Compaction

Level 0

Level 1

Target: 1 GB

Level 2

Target: 10 GB

Level 3

Target: 100 GB
Level-Based Compaction

Level 0

Level 1

Level 2

Level 3

Target: 100 GB

Target: 10 GB

Target: 1 GB
Level-Based Compaction

Level 1

Level 2

Level 3

Target: 100 GB

Target: 10 GB

Target: 1 GB
Level-Based Compaction

Level 1
Target: 1 GB

Level 2
Target: 10 GB

Level 3
Target: 100 GB
Level-Based Compaction

Level 1
- File
- File
- File

Target: 1 GB

Level 2
- File
- New File
- New File
- New File
- ……
- File
- File
- File

Target: 10 GB

Level 3
- File
- File
- File
- File
- File
- File
- File
- ……
- File
- File
- File

Target: 100 GB
Level-Based Compaction

Level 1

Target: 1 GB

Level 2

Target: 10 GB

Level 3

Target: 100 GB
Level-Based Compaction

Level 0

Level 1

Target: 1 GB

Level 2

Target: 10 GB

Level 3

Target: 100 GB
Why Log-Structure Merge Tree?
Measurements of MySQL+InnoDB Hosts’ Actual Resource Usages

- Read IOPS: < 5%
- Write IOPS: < 5%
- Peak Write Bandwidth: < 15%
- CPU: < 20%
- Write Endurance: last more than 3 years.
- *Space is the bottleneck*
MySQL + InnoDB vs MySQL + RocksDB

DB Size (Relative)

Bytes Written (Relative)
How do we estimate space efficiency in LSM?

Write Buffer in Memory

Persistent Store

10% Extra Space

Size Similar to User Data Size
Challenges
Challenges

- How to Keep LSM structure in good shape
- Chunk tombstone problem
- Bloom filter for range queries
Challenge 1: How to Keep LSM Tree in Good Shape?
Importance of Keeping LSM Tree In Good Shape

- Worse Case Space Efficiency
- Memory Caching
Our Solution:
Dynamic Level Size Target

Level 0
- File

Level 1
- File

Level N-2
- File
- ・・・
- Target: 8.76 GB < 1%

Level N-1
- File
- File
- ・・・
- File
- Target: 87.6 GB 9%

Level N
- File
- File
- File
- ・・・
- File
- 876 GB 90% of total size
Can we do better?

Level 0

Level 1

Level N-2

Level N-1

Level N

Size of level 1 can change by 10X when database size changes
Challenge 2: Bloom Filter for Range Queries
Bloom Filter

Not Found

Read Value

Filter out

Read Value

Bloom

Bloom

Bloom

Bloom

Bloom
Bloom Filter cannot be used in Range Queries

<table>
<thead>
<tr>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>......</td>
</tr>
<tr>
<td>Apple2013</td>
</tr>
<tr>
<td>Apple2015</td>
</tr>
<tr>
<td>Banana2012</td>
</tr>
<tr>
<td>Lemon2012</td>
</tr>
<tr>
<td>Lemon2013</td>
</tr>
<tr>
<td>Lemon2014</td>
</tr>
</tbody>
</table>

- Get ("Cherry2013") can use bloom filter
Trick: prefix bloom

<table>
<thead>
<tr>
<th>Keys</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>......</td>
</tr>
<tr>
<td>Apple</td>
<td>2013</td>
</tr>
<tr>
<td>Apple</td>
<td>2015</td>
</tr>
<tr>
<td>Banana</td>
<td>2012</td>
</tr>
<tr>
<td>Lemon</td>
<td>2012</td>
</tr>
<tr>
<td>Lemon</td>
<td>2013</td>
</tr>
<tr>
<td>Lemon</td>
<td>2014</td>
</tr>
<tr>
<td></td>
<td>......</td>
</tr>
</tbody>
</table>

- Define fruit part as “prefix”
- Can use bloom filter in range query: [cherry2010, cherry2015]
Open question: can we do better?

- Fruit part as “prefix”
- Boom filter not useful for range: [Banana2014, Banana2015]
- Bloom filter not useful for range: [C, H]
Challenge 3: Chunk of Tombstones
Deletions in LSM

Why do we have tombstones?

..., Put(1), Put(2), Put(3), Put(4), Put(5), Put(6), ...
Deletions in LSM

Why do we have tombstones?

... Delete(3), ...

... Put(1), Put(2), Put(3), Put(4), Put(5), Put(6), ...
When do we clear tombstones?

..., Delete(3), ...

Compact

..., Put(1), Put(2), Put(3), Put(4), Put(5), Put(6), ...

..., Delete(3), ...
What is tombstone?

..., Delete(3), ...

..., Put(1), Put(2), Put(3), Put(4), Put(5), Put(6), ...

Compact

..., Put(1), Put(2), Put(4), Put(5), Put(6), ...
Problem of Chunk Tombstones

..., Delete(1000), Delete(1001), ⋯, Delete(1999), ...

..., Put(999), Put(1000), Put(1001), Put(1002) ⋯, Put(1999), Put(2000), ...

• Range Query (1000, 2000) needs go through 2000 keys internally.
Our Trick: detect chunk tombstones and schedule compaction

Detect chunk deletion when generating the file, schedule compaction
Our Trick: detect chunk tombstone and schedule compaction
Our Trick: detect chunk deletion and schedule compaction

Detect chunk deletion when generating the file, schedule compaction
Our Trick: detect chunk tombstones and schedule compaction
Our Trick: detect chunk deletion and schedule compaction
Remaining Challenges

- Introduce more disk writes for some workloads
- Slow before compaction finishes
- Tombstones in mem tables.
Recap

- RocksDB Uses Log-Structure-Merge tree for space efficiency
- Keep LSM structure in good shape
- Chunk tombstone detection
- Prefix Bloom filter for range queries
Open Challenges

http://rocksdb.org