Prefix hashing in RocksDB

Speeding up queries for special workloads

Agenda

1. Why prefix hash?
2. What is prefix hash?
3. Implementations

Why Prefix Hash?

We want something faster than binary search

- RocksDB lookup is done by multiple binary searches
- When CPU is the bottleneck, we want binary search or hash lookup makes big difference in performance

When can binary search be avoided?

Example 1: Get() only queries

When can binary search be avoided?

Example 2: User activity logs

- Key: (user_id, activity timestamp)
- Query: find all activities done by a user between [t1, t2]

When can binary search be avoided?

Example 3: Page-like

- Key = (user_id, page_id), representing the user liked the page.
- Query: find all the pages that a user liked

When do we care about the difference?

- When storage is very fast, like RAM.
- When block cache hit rate is very high.
- When CPU usage is critical.
- When complicated key comparator is used.

What is prefix hash? The API

Prefix seek

Keys
“User date”
alice 20130325
alice 20130327
apple 20130320
frank 20130225
frank 20130320
frank 20130326

```
options.prefix_extractor.reset(NewFixedPrefixTransform(5));
options.memtable_factory.reset(NewHashSkipListRepFactory());
options.table_factory.reset(NewPlainTableFactory());
// Open DB with options

read_options.prefix_seek = true;
Iterator* iter = db->NewIterator(read_options);

// Print all the activities of Frank since 2013-03-01
for(iter->Seek(“frank 20130301”;
      iter->status().ok() && iter->Valid() && iter->key().startswith(“frank”);
      iter->Next()) {
  Print(iter->Key());
}
// print out “frank 20130320” and “frank 20130326”

// Print all the activities of Carol
for(iter->Seek(“carol”;
      iter->status().ok() && iter->Valid() && iter->key().startswith(“carol”);
      iter->Next()) {
  Print(iter->Key(), iter->Value();
}
// print out nothing
```

Prefix Iterating (What you cannot do)

Keys
“User date”
alice 20130325
alice 20130327
apple 20130320
frank 20130225
frank 20130320
frank 20130326

```
// Print all the activities of Users whose name starts with “a”
for(iter->Seek(“alice”;
      iter->status().ok() && iter->Valid() && iter->key().startswith(“a”);
      iter->Next()) {
  Print(iter->Key());
}
```

```
// Print all the keys
for(iter->SeekToFirst();
      iter->status().ok() && iter->Valid();
      iter->Next()) {
  Print(iter->Key());
}
```

Implementations

Choose from Memtable and SST implementations

Different Implementations

- Mem tables
  - Prefix skip list
  - Prefix linked list
- SST tables
  - Block based table (default SST format) with hash index
  - “PlainTable” – an SST format optimized for memory-only use cases, where rows are only addressed by offsets.

Conclusions

- Use prefix hash when:
  - Always query within a prefix
  - Want to use less CPU
- Prefix hash is easy to configure and use
- There are different choices of mem table and SST table implementations that support prefix hash

Thank you!