COMMUNITY
THE ASF CONFERENCE
CODE

Page Against the Machine

Thrift, CQL, Bytes, and Tombstones

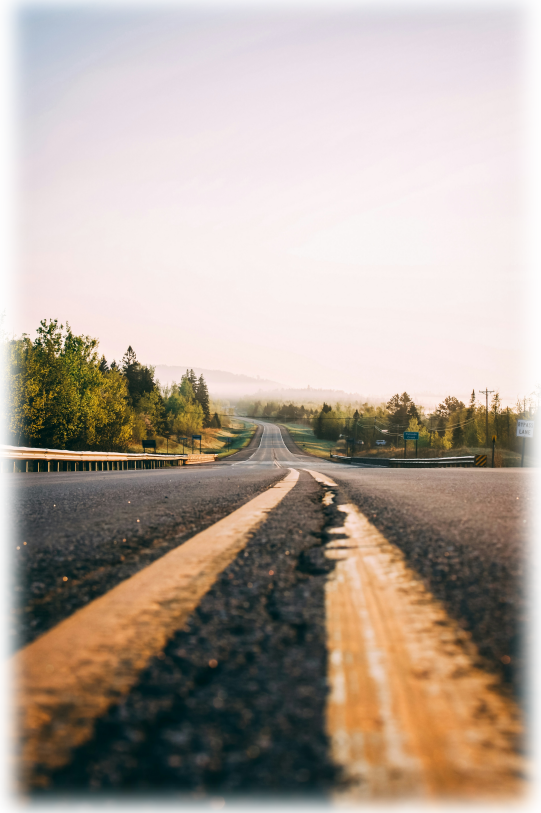
Josh McKenzie
Apache Cassandra Committer, PMC
Apple Inc.

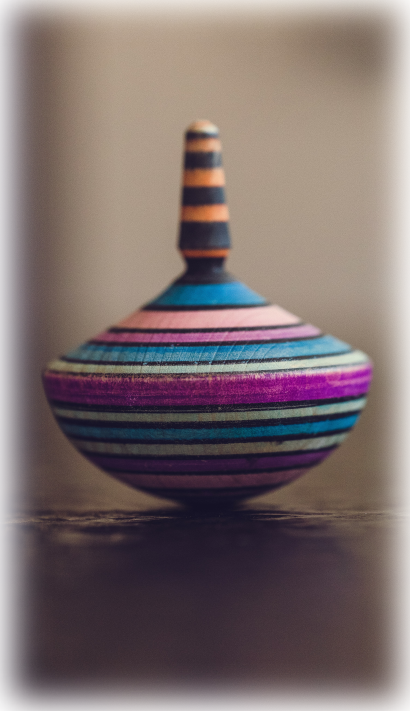
Our Paging Journey

Too Thrifty for our own good

- Inception
- From Thrift to CQL
- By rows or bytes?
- Tombstones
- The Deep Future?

We are *still here*





Inception

“An idea. Resilient... highly contagious. Once an idea has taken hold of the brain it's almost impossible to eradicate.” ~Cobb

Paging in Thrift: slicing. Unfriendly slicing.

- * A SlicePredicate is **similar to a mathematic predicate (see http://en.wikipedia.org/wiki/Predicate_(mathematical_logic))**, which is described as "a property that the elements of a set have in common." SlicePredicate's in Cassandra are described with either a list of column_names or a SliceRange. If column_names is specified, slice_range is ignored.
- public class SlicePredicate implements org.apache.thrift.TBase<SlicePredicate, SlicePredicate._Fields>

- /** A slice range is a structure that stores basic range, ordering and limit information for a query that will return multiple columns. It could be thought of as Cassandra's version of LIMIT and ORDER BY **/
- public class SliceRange implements org.apache.thrift.TBase<SliceRange, SliceRange._Fields>

Paging in Thrift: Slices

```
private static List<KeySlice> getNextPage(Cassandra.Client client, ByteBuffer startKey, ByteBuffer endKey)
            throws TException {

ColumnParent columnParent = new ColumnParent(COLUMN_FAMILY);

SliceRange sliceRange = new SliceRange();
sliceRange.setStart(new byte[START_BYTE]);
sliceRange.setFinish(new byte[END_BYTE]);
sliceRange.setCount(1000);  // This is the column count, not the key count

SlicePredicate slicePredicate = new SlicePredicate();
slicePredicate.setSlice_range(sliceRange);

KeyRange keyRange = new KeyRange();
keyRange.setStart_key(startKey);
keyRange.setEnd_key(endKey);
keyRange.setCount(PAGE_SIZE);

return client.get_range_slices(columnParent, slicePredicate, keyRange, ConsistencyLevel.ONE);
```

Paging in Thrift: iteration

```
while (hasMore) {
  List<KeySlice> pageSlices = getNextPage(client, lastKey, endKey);

  if (pageSlices.isEmpty()) {
    hasMore = false;
  }
  else {
    allSlices.addAll(pageSlices);
    lastKey = ByteBuffer.wrap(pageSlices.get(pageSlices.size() - 1).getKey());

    if (compareByteBuffers(lastKey, endKey) >= 0) {
      hasMore = false;
    }
  }
}
```

Paging in the Native Protocol (CQL)

- CASSANDRA-4415: Add cursor API/auto paging to the native CQL protocol”
- June 2013; Sylvain Lebresne with Aleksey Yeschenko on review
- Introduced the paging you know and love today
- Addition of server-side split up of user query (instead of client-side supplied)
- **From Cell to Row**

How CQL Paging Works

- Client asks for results of a query
- Server returns result_page_size + results ***+ has_more_pages*** + continuation (CK)
- User returns query w/CK for resuming next page
- ***Server dictates when things are done*** *(Excerpt from native_procotol_v3-v5.spec)*

Clients should not rely on the actual size of the result set returned to decide if there are more results to fetch or not. Instead, they should always check the Has_more_pages flag (unless they did not enable paging for the query obviously). Clients should also not assert that no result will have more than <result_page_size> results. While the current implementation always respects the exact value of <result_page_size>, we reserve the right to return slightly smaller or bigger pages in the future for performance reasons.

CQL Paging Examples

Synchronous paging over Rows in a ResultSet

```
ResultSet rs = session.execute(statement);
for (Row r : rs.all()) {…}
```

Asynchronous paging, checking paging state:

```
while (rs.getExecutionInfo().getPagingState() != null) {
  rs = rs.fetchMoreResults().get(5, TimeUnit.SECONDS);
  // Do Stuff
}
```

CQL Paging Limitations

- Highly variable data set sizes (a row is not a row is not a row)
- Initially, OOM's from too many tombstones
- Evolved to **TombstoneOverwhelmingException**
  - Added in CASSANDRA-6117 as an alternative to OOM'ing nodes
  - October 2013, Jonathan Ellis w/Sylvain Lebresne on review
  - Query aborts (instead of node detonates)
- Still static, user-supplied paging boundaries
  - See CASSANDRA-6492: "Have server pick query page size by default" (open 12/2013 and still unresolved)

Upcoming Paging Changes

‘If you do not change direction, you may end up where you are heading.’
~ Lao Tzu

Paging by Bytes: CASSANDRA-11745

IN REVIEW

- A new version of *setFetchSize()*?
- JIRA created 2016; patch dropped in 2023 by Jacek Lewandowski
- Pages by bytes; *also has LIMIT BYTES functionality in the patch*
- Implementation: ~ 20 unused flags in QueryOptions#Codec, adding **PAGE_SIZE_IN_BYTES** flag
- Resolves “non-homogeneous row returns make my pages unpredictable or timeout”

Paging by Bytes: The UX

- Limiting data for a single query:
  - CQL: SELECT * FROM ks.tbl LIMIT BYTES N;

- Limiting data for pages (work in progress):
  - 4.x driver: basic.request.page-size
    - basic.request.page-size-bytes?
  - 3.x driver:

```
Cluster cluster = Cluster.builder()
    .addContactPoint("127.0.0.1")
    .withQueryOptions(new QueryOptions().setFetchSize(2000))
    .build();
// Possibly…
.withQueryOptions(new QueryOptions().setFetchSizeBytes(4096))
```

Design: PageSize and Bytes

- PageSize.java

```
public final int rows;
public final int bytes;
```

New Concepts!

- Checking: MultiPartitionPager

```
// We are done if we have reached the page size or in the case
// of GROUP BY if the current pager is not exhausted.
boolean isDone = counted >= pageSize.rows
    || bytes >= pageSize.bytes
    || (result != null
        && limit.isGroupByLimit()
        && !pagers[current].isExhausted());
```

New Check!

Paging Across Tombstones (CASSANDRA-18424)

IN PROGRESS

- NativeProtocol_vN.spec make it clear we can...
  - Use the ClusteringKey as the last row where we hit our tombstone limit
  - Trust clients to pass that back to us as their paging key
  - Short-circuit pages when we hit a tombstone limit
- What does this solve?
  - TombstoneOverwhelmingExceptions!
  - Sort of. Long chains of empty page results are wasteful

How do you use it?

- Enabling should be completely transparent to users
  - Trade out TOE for smaller or even empty page results
- No client / user action required
- Per-node setting
- cassandra.yaml
  - `tombstone_warn_threshold: 1000`
  - `tombstone_failure_threshold: 100000`
  - `tombstone_paging_enabled: false`

Design: Stopping at Tombstone Boundaries

- SinglePartitionPager extends AbstractQueryPager
- SinglePartitionReadCommand extends ReadCommand; add:
  public @Nullable Row tombstoneLimitedRow() // ClusteringKey back to client
  - In our iterator: private UnfilteredPartitionIterator withMetricsRecording

```
if (isStopped())
    tombstoneLimitedRow = row;
```

- In StoppingTransformation:

```
/** Allows deriving classes to introspect and take action
if iteration has been stopped. */
protected boolean isStopped()
```

Design: Flagging we're Stopped

```
ReadCommand#withMetricsRecording:

private void countTombstone(ClusteringPrefix clustering)

if (throwOnTombstoneLimit) {
  …
  throw new TombstoneOverwhelmingException(tombstones, query,
      ReadCommand.this.metadata(), currentKey, clustering);
}
else {
  …
  stopInPartition(); // New method in StoppingTransformation
}
```

Design: Making the Pager Aware

- Hold a temporary cache of ReadCommands in our pager

```
AbstractQueryPager.onClose():
  for (ReadQuery cmd : pageQueries)
  {
      Row tsRow = cmd.tombstoneLimitedRow();
      if (tsRow != null)
      {
          lastRow = cmd.tombstoneLimitedRow();
          assert lastRow != null;
          hitTombstoneLimit = true;
          recordLast(lastKey, lastRow);
          break;
      }
  }
```

PagingAcrossTombstonesTest.java: Basic Unit Tests

- 26 base tests, 104 total with configuration
- 4 configurations
  - Sync forward
  - Sync backward
  - Async forward
  - Async backward

AbstractPagingValidator: Model, Generation, and Validation

- Stakes are high; functional data loss at client is our risk
- Harry fuzzes (at page size 1)
- AbstractPagingValidator.java: model of Cells, liveness, tombstones
- Paired generation and deletion with model and CQL queries
  - generate[Linear|Randomized](), deleteRandom[Cell|Row](), deleteRandomized()
- validateCellState(): Compare expectedCellState to seenCellState

```
protected final Map<Integer, CellState> seenCellState = new HashMap<>();
```

- long runSeed provided on failure

PagingFuzzTest.java: Randomize Bounds

Randomize parameters per run:

```
    private void randomizeForNextTest()
    {
        int pageSize = randomFromRange(1, 10000);
        int failureThreshold = randomFromRange(1, 10000);
        validator.beforeTest();
        validator.prepareTest(pageSize, failureThreshold);
    }
```

PagingFuzzTest.java: Our Commands

- Commands:
  - GenerateCommand: linear or random
  - DeleteCommand: [linear|random][cell|row][range]
  - StateCommand: Reversed, Static
- `private List<PagingFuzzCommand> generateCommands() {}`
  - Randomized add weight vs. delete weight
  - Run against 10–50 commands
  - 10% chance of StateCommand

PagingFuzzTest.java: Running a Test

```
try {
    validator.reset();
    randomizeForNextTest();
    reproCommandHistory = populateData();
    validator.runTest();
}
catch (AssertionError ae) {
    // Print out command to run from terminal to reproduce
    // including seed. Also generate .java unit test that
    // will directly reproduce if copy/pasted into a test.
}
```

What's left?

- Cluster dtests
  - Mix of old and new nodes with and without feature on
  - Mix of new nodes w/all feature off, with all feature on
- Consider what to do about RTBoundCloser.java (CASSANDRA-14515)
  - Issues if we send a response w/out closing bound; data loss
  - Current impl: Don't short-circuit on tombstones if within a RT
    - Could lead to long paging before returning to client
- Review

The Future

“I’ve always been more interested in the future than in the past.”
~ Grace Hopper

In a the near future, you will be able to page by bytes and gracefully page over tombstones in Cassandra.

UX: Revolution, not Evolution?

A Theory of responsibility

- Beginning: Users owned entirety of paging
- Middle (now): Users own constraints (rows)
- End: Users express *intent:*
  - “At most, this much data”
  - “At most, take this long”
- Signals from paging feeding into LSM maintenance strategies

To Recap

- We still page like it was Thrift
- Paging hasn’t changed much in 7 years
- Paging is about to change a lot in good ways
- There is still more work to do!

Q&A