Cassandra 2012:
What's New & Upcoming

Sam Tunnicliffe

DATASTAX

- sam@datastax.com

- DSE : integrated Big Data platform
  - Built on Cassandra
  - Analytics using Hadoop (Hive/Pig/Mahout)
  - Enterprise Search with Solr

Cassandra in 2012

- Cassandra 1.1 – April 2012
- Cassandra 1.2 – November 2012
- What's on the roadmap

Release Schedule

Version	Release Date
0.5	24 Jan 2010
0.6	13 April 2010
0.7	9 January 2011
0.8	2 June 2011
1.0	18 October 2011
1.1	24 April 2012
1.2	End November 2012 ?

- Currently working to six month release cycle
- Minor releases as and when necessary

Global Row & Key Caches

- Prior to 1.1 separate row & key caches per column family
- Since 1.1 single row cache & single key cache shared across all column families
- Simpler configuration
  - Globally: `{key,row}_cache_size_in_mb`
  - Per CF: `ALL|NONE|KEYS_ONLY|ROWS_ONLY`

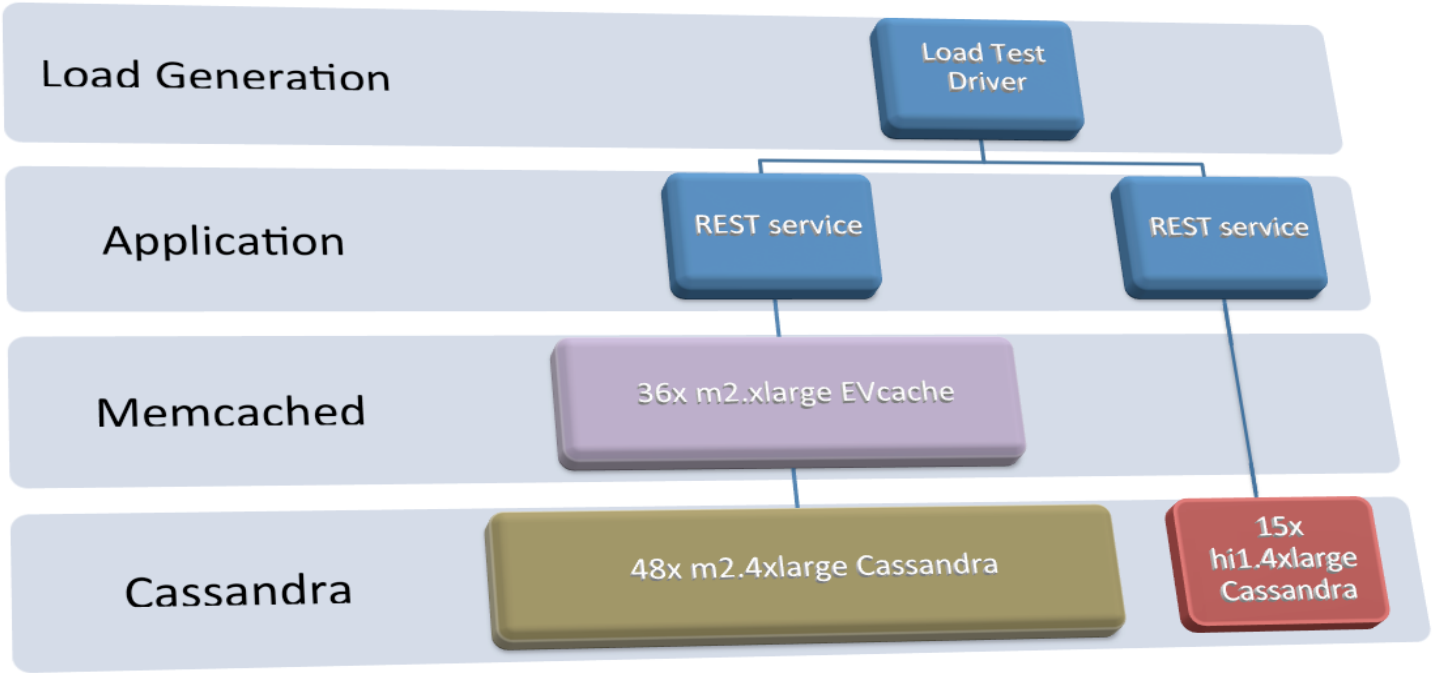
More Granular Storage Config

- Pre 1.1 one storage location per keyspace
  - `/var/lib/cassandra/ks/cf-hc-1-Data.db`
- In 1.1 one location per column family
  - `/var/lib/cassandra/ks/cf/ks-cf-hc-1-Data.db`
- Allows you to pin certain data to particular storage system

Why?

Cassandra Disk vs. SSD Benchmark

Same Throughput, Lower Latency, Half Cost



NETFLIX

http://techblog.netflix.com/2012/07/benchmarking-high-performance-io-with.html

Why?

http://www.slideshare.net/rbranson/cassandra-and-solid-state-drives

Row Level Isolation

- Batched writes within row have always been atomic
- Batched writes within row are now also isolated

Row Level Isolation

- So Writing

```
UPDATE foo
SET column_x='a' AND column_y='1'
WHERE id='bar';
```

- Followed by

```
UPDATE foo
SET column_x='b' AND column_y='2'
WHERE id='bar';
```

- No reader ever sees either

```
{column_a:'a', column_b:'2'}
{column_x:'b', column_y:'1'}
```

Misc Other Stuff

- Windows off heap cache
- write survey mode
- commitlog segment pre-allocation/recycling
- abortable compactions
- multi threaded streaming
- Hadoop improvements
  - utilise secondary indexes from Hadoop
  - better wide row support

CQL 3.0

- Beta in 1.1, finalized in 1.2
- Motivations
  - Better support for wide rows
  - Native syntax for composites

CQL 3.0

- Wide Rows
- Composites
- Transposition

CQL 3.0

```
CREATE TABLE comments (
  id text,
  posted_at timestamp,
  author text,
  content text,
  karma int,
  PRIMARY KEY( id, posted_at )
);
```

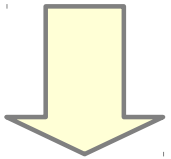
```
INSERT INTO comments (id, posted_at, author, content, karma)
VALUES ('article_x', '2012-10-31T08:00:00', 'freddie', 'awesome', 100);

INSERT INTO comments (id, posted_at, author, content, karma)
VALUES ('article_x', '2012-10-31T09:59:59', 'rageguy', 'F7U12', 0);
```

CQL 3.0 Transposition

```
{ article_x : { 2012-10-31T08:00:00 + author  => freddie }
              { 2012-10-31T08:00:00 + content => awesome }
              { 2012-10-31T08:00:00 + karma   => 100 }

              { 2012-10-31T09:59:59 + author  => rageguy }
              { 2012-10-31T09:59:59 + content => F7U12 }
              { 2012-10-31T09:59:59 + karma   => 1 }
}
{ article_y : { 2012-10-28T15:30:12 + author...
```



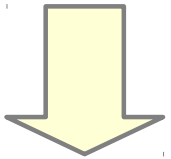
```
cqlsh:hn> select * from comments where id = 'article_x';

 id        | posted_at            | author  | content | karma
-----------+----------------------+---------+---------+-------
 article_x | 2012-10-31 08:00:00 | freddie | awesome |   100
 article_x | 2012-10-31 09:59:59 | rageguy |   F7U12 |     1
```

CQL 3.0 Transposition

```
{ article_x : { 2012-10-31T08:00:00 + author  => freddie }
              { 2012-10-31T08:00:00 + content => awesome }
              { 2012-10-31T08:00:00 + karma   => 100 }

              { 2012-10-31T09:59:59 + author  => rageguy }
              { 2012-10-31T09:59:59 + content => F7U12 }
              { 2012-10-31T09:59:59 + karma   => 1 }
}
{ article_y : { 2012-10-28T15:30:12 + author...
```



```
cqlsh:hn> select * from comments where id = 'article_x'
          and posted_at <= '2012-10-31T08:00:00' ;

 id         | posted_at            | author  | content | karma
------------+----------------------+---------+---------+-------
 article_x | 2012-10-31 08:00:00 | freddie | awesome |   100
```

CQL 3.0 Collections

- Typed collections – Set, Map, List
- Good fit for denormalizing small collections
- Less ugly, more efficient than previous approach
- Composites under the hood
- Each element stored as one column
  - So each can have an individual TTL

CQL 3.0 Sets

- Collection of typed elements
- No duplicate elements
- Actually a sorted set
  - Ordering determined by natural order elements

CQL 3.0 Sets

```
CREATE TABLE users (
          user_id text PRIMARY KEY,
          first_name text,
          last_name text,
          emails set<text>
      );
```

- Set entire value with set literal

```
INSERT INTO users (user_id, first_name, last_name, emails)
VALUES('sam', 'Sam', 'Tunnicliffe',
       {'sam@beobal.com', 'sam@datastax.com'});
```

CQL 3.0 Sets

- Insert and remove

```
UPDATE users
SET emails = emails + {'sam@cohodo.net'}
WHERE user_id = 'sam';

UPDATE users
SET emails = emails – {'sam@cohodo.net'}
WHERE user_id = 'sam'
```

- No distinction between empty and null set

```
UPDATE users
SET emails = {}
WHERE user_id = 'sam';

DELETE emails
FROM users
WHERE user_id = 'sam';
```

CQL 3.0 Lists

- Ordering determined by user, not by natural ordering of elements
- Allows multiple occurrences of same value

CQL 3.0 Lists

```
ALTER TABLE users ADD todo list<text>;
```

- Set, Prepend & Append

```
UPDATE users
SET todo = [ 'get to Sinsheim', 'give presentation' ]
WHERE user_id = 'sam';

UPDATE users
SET todo = [ 'finish slides' ] + todo
WHERE user_id = 'sam';

UPDATE users
SET todo = todo + [ 'drink beer' ]
WHERE user_id = 'sam';
```

CQL 3.0 Lists

- Access elements by index
  - Less performant, requires read of entire list

```
UPDATE users
SET todo[2] = 'slow down'
WHERE user_id = 'sam';

DELETE todo[3]
FROM users
WHERE user_id = 'sam';
```

- Remove by value

```
UPDATE users
SET todo = todo – ['finish slides']
WHERE user_id = 'sam';
```

CQL 3.0 Maps

- Collection of typed key/value pairs
- Like sets, maps are actually ordered
  - Ordering determined by the type of the keys
- Operate on entire collection or individual element by key

CQL 3.0 Maps

```
ALTER TABLE users ADD accounts map<text, text>;
```

- Set entire value with map literal

```
UPDATE users
SET accounts = { 'twitter' : 'beobal',
                 'irc'     : 'sam___' }
WHERE user_id = 'sam';
```

- Access elements by key

```
UPDATE users
SET accounts['irc'] = 'beobal@freenode.net'
WHERE user_id = 'sam';

DELETE accounts['twitter']
FROM users
WHERE user_id = 'sam';
```

CQL 3.0 Collections

- Can only retrieve a collection in its entirety
  - Collections are not intended to be very large
  - Not a replacement for a good data model
- Typed but cannot currently be nested
  - Can't define a `list<list<int>>`
- No support for secondary indexes
  - On the roadmap but not yet implemented

CQL 3.0 Binary Protocol

- Motivations
  - Ability to optimize for specific use-cases
  - Reduce client dependencies
  - Lay groundwork for future enhancements such as streaming/paging

CQL 3.0 Binary Protocol

- Framed protocol, designed natively for CQL 3.0
- Built on Netty
- Java, Python & C drivers in development
- Switched off by default in 1.2
- Thrift API will remain, no breaking API changes

Virtual Nodes

- Evolution of Cassandra's distribution model
- Nodes own multiple token ranges
- Big implications for operations
  - Replacing failed nodes
  - Growing / shrinking cluster
- Stay put for Eric

Concurrent Schema Changes

- Online schema changes introduced in 0.7
- Initially schema changes needed serializing
  - Update schema
  - Wait for propagation before making next change
- Mostly fixed in 1.1
- Completed in 1.2
  - Concurrent ColumnFamily creation

Better JBOD Support

- Growing data volumes and compaction requirements means moar disk
- Pre 1.2 best practice is to run RAID10
- Improvements to handling disk failures
  - `disk_failure_policy : stop | best_effort | ignore`
  - `best_effort` implications for writes / reads
- Better write balancing for less contention

Misc Other Stuff

- Query Tracing
- Speedup slicing wide rows from disk
- Atomic Batches

Future Plans

- Improved caching for wide rows
- Remove 2-phase compaction
- Remove SuperColumns internally

Thank you