Apache Cassandra in Action

Jonathan Ellis
jbellis@datastax.com / @spyced
Why Cassandra?

• Relational databases are not designed to scale
• B-trees are slow
  - and require read-before-write
December, 2002

(“The eBay Architecture,” Randy Shoup and Dan Pritchett)
The Log-Structured Merge-Tree, Bigtable: A Distributed Storage System for Structured Data
Cassandra in production

- Digital Reasoning: NLP + entity analytics
- OpenWave: enterprise messaging
- OpenX: largest publisher-side ad network in the world
- Cloudkick: performance data & aggregation
- SimpleGEO: location-as-API
- Ooyala: video analytics and business intelligence
- ngmoco: massively multiplayer game worlds
FUD?

- “Cassandra is only appropriate for unimportant data.”
• Write to commitlog
  - fsync is cheap since it’s append-only
• Write to memtable
• [amortized] flush memtable to sstable
SSTable format, briefly

Sorted [clustered] by row key
Scaling
(A-L)
Key “C”
Reliability

- No single points of failure
- Multiple datacenters
- Monitorable
Some headlines

- “Resyncing Broken MySQL Replication”
- “How To Repair MySQL Replication”
- “Fixing Broken MySQL Database Replication”
- “Replication on Linux broken after db restore”
- “MySQL :: Repairing broken replication”
Good architecture solves multiple problems at once

- Availability in single datacenter
- Availability in multiple datacenters
Key "C"
Tuneable consistency

• ONE, QUORUM, ALL
• R + W > N
• Choose availability vs consistency (and latency)
Monitorable
OpsCenter

Cluster Views:
- Ring View
- Table View

Keyspaces:
- MyKeyspace
  - CF1
- ColumnFamily2
  - MyKeyspace 2

Color Key:
- Normal Load
- Moderate Load
- High Load
- Unresponsive

Node 3C2C1AA6
- Active
- Size: 214 GB
- Load: 1.2
- IP: 127.0.0.1
When do you need Cassandra?

- Ian Eure: “If you’re deploying memcache on top of your database, you’re inventing your own ad-hoc, difficult to maintain NoSQL data store”
Curt Monash: “**ACID-compliant transaction integrity** commonly costs more in terms of DBMS licenses and many other components of TCO (Total Cost of Ownership) than [scalable NoSQL]. Worse, it **can actually hurt application uptime**, by forcing your system to pull in its horns and stop functioning in the face of failures that a non-transactional system might smoothly work around. Other flavors of “complexity can be a bad thing” apply as well. Thus, **transaction integrity can be more trouble than it’s worth.**” [Curt’s emphasis]
Keyspaces & ColumnFamilies

- Conceptually, like “schemas” and “tables”
Inside CFs, columns are dynamic

- Twitter: “Fifteen months ago, it took two weeks to perform ALTER TABLE on the statuses [tweets] table.”
ColumnFamilies

• **Static**
  - Object data

• **Dynamic**
  - Precalculated query results
“static” columnfamilies

Users

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
<th>Name</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>zznate</td>
<td>*</td>
<td>Nate</td>
<td></td>
</tr>
<tr>
<td>driftx</td>
<td>*</td>
<td>Brandon</td>
<td></td>
</tr>
<tr>
<td>thobbs</td>
<td>*</td>
<td>Tyler</td>
<td></td>
</tr>
<tr>
<td>jbellis</td>
<td>*</td>
<td>Jonathan</td>
<td>riptano.com</td>
</tr>
</tbody>
</table>
"dynamic" columnfamilies

Following

- zznate
- driftx: thobbs:
- thobbs: zznate:
- jbellis driftx: mdennis: pcmanus thobbs: xedin: zznate
Inserting

- Really “insert or update”
- Not a key/value store – update as much of the row as you want
Example: twissandra

- http://twissandra.com
CREATE TABLE users (  
id INTEGER PRIMARY KEY,  
username VARCHAR(64),  
password VARCHAR(64)  
);

CREATE TABLE following (  
user INTEGER REFERENCES user(id),  
followed INTEGER REFERENCES user(id)  
);

CREATE TABLE tweets (  
id INTEGER,  
user INTEGER REFERENCES user(id),  
body VARCHAR(140),  
timestamp TIMESTAMP  
);
create column family **users** with comparator = UTF8Type and **column_metadata** = [{column_name: password, validation_class: UTF8Type}]

create column family **tweets** with comparator = UTF8Type and **column_metadata** = [{column_name: body, validation_class: UTF8Type}, {column_name: username, validation_class: UTF8Type}]

create column family **friends** with comparator = UTF8Type
create column family **followers** with comparator = UTF8Type

create column family **userline** with comparator = LongType and **default_validation_class** = UUIDType
create column family **timeline** with comparator = LongType and **default_validation_class** = UUIDType
Connecting

CLIENT = pycassa.connect_thread_local('Twissandra')
USER = pycassa.ColumnFamily(CLIENT, 'User')
RowKey: ericflo
=> (column=password, value=****, timestamp=1289446382541473)

-------------------
RowKey: jbellis
=> (column=password, value=****, timestamp=1289446438490709)

uname = 'jericevans'
password = '**********'
columns = {'password': password}
USER.insert(uname, columns)
Natural keys vs surrogate
Friends and Followers

RowKey: ericflo

=> (column=jbellis, value=1289446467611029, timestamp=1289446467611064)

=> (column=b6n, value=1289446467611031, timestamp=1289446467611080)

to_uname = 'ericflo'

FRIENDS.insert(uname, {to_uname: time.time()})
FOLLOWERS.insert(to_uname, {uname: time.time()})
Tweets

RowKey: 92dbeb50-ed45-11df-a6d0-000c29864c4f

=> (column=body, value=Four score and seven years ago, timestamp=1289446891681799)

=> (column=username, value=alincoln, timestamp=1289446891681799)

-------------------

RowKey: d418a66e-edc5-11df-ae6c-000c29864c4f

=> (column=body, value=Do geese see God?, timestamp=1289501976713199)

=> (column=username, value=pdrome, timestamp=1289501976713199)
RowKey: ericflo

=> (column=1289446393708810, value=6a0b4834-ed44-11df-bc31-000c29864c4f, timestamp=1289446393710212)

=> (column=1289446397693831, value=6c6b5916-ed44-11df-bc31-000c29864c4f, timestamp=1289446397694646)

=> (column=1289446891681780, value=92dbeb50-ed45-11df-a6d0-000c29864c4f, timestamp=1289446891685065)

=> (column=1289446897315887, value=96379f92-ed45-11df-a6d0-000c29864c4f, timestamp=1289446897317676)
Userline

**zznate**
- 1289847840615: 3f19757a-c89d...
- 128984784425: 844e75e2-b546...

**driftx**
- 1289847887086: a20fcf52-595c...

**thobbs**
- 1289847887086: a20fcf52-595c...

**jbellis**
- 1289847840615: 3f19757a-c89d...
- 128984784425: 844e75e2-b546...
@daveauth you read NEWS.txt yes?
about 10 hours ago via web in reply to daveauth

zzzeeek SQLAlchemy 0.6.0 is released. http://bit.ly/cLot6o sqlalchemy
7:03 PM Apr 18th via Tweetie
Retweeted by you and 13 others

standardssociety cf--cassandra (Active): A CFX wrapper that allows Coldfusuion to interact with Apache Cassandra
12:28 PM Apr 16th via twitterfeed
Retweeted by you

dberlind John Quinn (VP Eng, Digg) @ #UTR: If you want a database that really scales, don't use Oracle, PostgreSQL. Use Cassandra
11:31 AM Apr 16th via TweetDeck
Retweeted by you and 6 others

rk slides from my #cassandra presentation yesterday: http://www.slideshare.net/ryansking/scaling-twitter-with-cassandra chirp
3:31 PM Apr 16th via Tweetie
Retweeted by you and 8 others

woothshirt $10.00 :@--: LAST CALL http://shirt.woot.com
about 2 hours ago via web

klein_stephane Est-ce qu'il est possible de lancer des requêtes SPARQL sur Amazon ? exemple, avoir les titres tous les livres de l'auteur "foobar"
about 2 hours ago via web

nitot Superbe : Quand Socrate nous aide à mieux comprendre le logiciel libre: http://tinyurl.com/y6ehhqe
about 2 hours ago via Identica
Retweeted by klein_stephane and 9 others
Timeline

RowKey: ericflo

=> (column=1289446393708810, value=6a0b4834-ed44-11df-bc31-000c29864c4f, timestamp=1289446393710212)

=> (column=1289446397693831, value=6c6b5916-ed44-11df-bc31-000c29864c4f, timestamp=1289446397694646)

=> (column=1289446891681780, value=92dbeb50-ed45-11df-a6d0-000c29864c4f, timestamp=1289446891685065)

=> (column=1289446897315887, value=96379f92-ed45-11df-a6d0-000c29864c4f, timestamp=1289446897317676)
Adding a tweet

tweet_id = str(uuid())
body = '@ericflo thanks for Twissandra, it helps!'
timestamp = long(time.time() * 1e6)

columns = {'uname': useruuid, 'body': body}
TWEET.insert(tweet_id, columns)

columns = {ts: tweet_id}
USERLINE.insert(uname, columns)

TIMELINE.insert(uname, columns)
for follower_uname in FOLLOWERS.get(uname, 5000):
    TIMELINE.insert(follower_uname, columns)
Reads

timeline = USERLINE.get(uname, column_reversed=True)
tweets = TWEET.multiget(timeline.values())

start = request.GET.get('start')
limit = NUM_PER_PAGE

timeline = TIMELINE.get(uname, column_start=start, column_count=limit, column_reversed=True)
tweets = TWEET.multiget(timeline.values())
Programatically

- Don't use thrift directly
- Higher level clients have a lot of features you want
  - Knowledge about data types
  - Connection pooling
  - Automatic retries
  - Logging
def get_client(host='127.0.0.1', port=9170):
    socket = TSocket.TSocket(host, port)
    transport = TTransport.TBufferedTransport(socket)
    transport.open()
    protocol =
    TBinaryProtocol.TBinaryProtocolAccelerated(transport)
    client = Cassandra.Client(protocol)
    return client
data = {'id': useruuid, ...}
columns = [Column(k, v, time.time())
    for (k, v) in data.items()]
mutations = [Mutation(ColumnOrSuperColumn(column=c))
    for c in columns]
rows = {useruuid: {'User': mutations}}

client.batch_mutate('Twissandra', rows, ConsistencyLevel.ONE)
API layers

- libpq
- JDBC
- JPA
- Thrift
- Hector
- Hector object-mapper
Running twissandra

- Login: notroot/notroot
  - (root/riptano)

- cd twissandra

- python manage.py runserver &

- Navigate to http://127.0.0.1:8000

- Login as jim/jim, tom/tom, or create your own
One more thing

- !PUBLIC! userline
Exercise 1

- $ cassandra-cli --host localhost
- ] use twissandra;
- ] help;
- ] help list;
- ] help get;
- ] help del;

- Delete the most recent tweet
  - How would you find this w/o looking at the UI?
Exercise 2

- User jim is following user tom, but twissandra doesn't populate Timeline with tweets from before the follow action.
- Insert a tweet from tom before the follow action into jim's timeline
Secondary (column) indexes
Exercise 3

- Add a state column to the Tweet column family definition, with an index (index_type KEYS).
  - Hint: a no-op update column family on Tweet would be update column family Tweet with column_metadata=[{column_name:body, validation_class:UTF8Type}, {column_name:username, validation_class:UTF8Type}]

- Set the state column on several tweets to TX. Select them using get ... where.
Language support

- **Python**
  - pycassa
  - telephus

- **Ruby**
  - Speed is a negative

- **Java**
  - Hector

- **PHP**
  - phpcassa
Done yet?

- Still doing 1+N queries per page
- Solution: Supercolumns
Applying SuperColumns to Twissandra

**jbellis**

1289847840615

- **Id:** 3f19757a-c89d...
- **uname:** zznate
- **body:** O stone be not so

1289847844275

- **Id:** 844e75e2-b546...
- **uname:** driftx
- **body:** Rise to vote sir

1289847887086

- **Id:** a20fcf52-595c...
- **uname:** zznate
- **body:** I prefer pi
Supercolumns: limitations

- Requires reading an entire SC (not the entire row) from disk even if you just want one subcolumn
UUIDs

• Column names should be uuids, not longs, to avoid collisions

• Version 1 UUIDs can be sorted by time ("TimeUUID")

• Any UUID can be sorted by its raw bytes ("LexicalUUID")
  - Usually Version 4
  - Slightly less overhead
What documents contain term X?
- ... and term Y?
- ... or start with Z?
Fields and Terms

<doc>
  <field name="title">apache talk</field>
  <field name="date">20110201</field>
</doc>

<table>
<thead>
<tr>
<th>field</th>
<th>term</th>
<th>freq</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>apache</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>title</td>
<td>talk</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>date</td>
<td>20110201</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
create column family `documents` with comparator = BytesType;

Create column family `terminfo` with column_type = `Super` and comparator = BytesType and subcomparator = BytesType;
## Lucandra data

<table>
<thead>
<tr>
<th>Document Key</th>
<th>col name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;documentId&quot;</td>
<td>=&gt;</td>
<td>{ fieldName, value }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term Key</th>
<th>col name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;field/term&quot;</td>
<td>=&gt;</td>
<td>{ documentId, position vector }</td>
</tr>
</tbody>
</table>
Lucandra queries

- get_slice
- get_range_slices
- No silver bullet
FAQ: counting

- UUIDs + batch process
- column-per-app-server
- counter API (after 1.0 is out)
Locking

- Zookeeper
- Cages: http://code.google.com/p/cages/
- Not suitable for multi-DC
<table>
<thead>
<tr>
<th>counter1</th>
<th>672e34a2-ba33...</th>
<th>b681a0b1-58f2...</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter2</td>
<td>3f19757a-c89d...</td>
<td>844e75e2-b546...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>counter1</th>
<th>aggregated: 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter2</td>
<td>aggregated: 42</td>
</tr>
<tr>
<td>Column per appserver</td>
<td>counter1</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>672e34a2-ba33: 12</td>
</tr>
<tr>
<td></td>
<td>b681a0b1-58f2: 4</td>
</tr>
<tr>
<td></td>
<td>1872c1c2-38f1: 9</td>
</tr>
</tbody>
</table>
Counter API

- key
- counter1: (14, 13, 9)
- counter2: (11, 15, 17)
General Tips

- Start with queries, work backwards
- Avoid storing extra “timestamp” columns
- Insert instead of check-then-insert
- Use client-side clock to your advantage
- use TTL
- Learn to love wide rows