Cassandra Persistence for Online Systems
What Actually Works

John Sumision
FamilySearch
Introduction

- Cassandra for online systems
- Introduction to Family Tree
- Event-sourced persistence model
- Surprises & Solutions
Cassandra for Online Systems

- KillrVideo from Datastax Academy
- Classic use cases (from 2014)
  - Product Catalog / Playlist
  - Recommendation Engine
  - Sensor Data/IOT
  - Messaging
  - Fraud Detection

https://www.datastax.com/2014/06/what-are-people-using-cassandra-for
Cassandra for Online Systems

- CQL-based schemas (record & fields)
- Blob-based schemas (JSON inside blob)
- Time-series schemas (sensor data)
- Event-sourced schemas (events & views)

Restrictions:
  - No joins
  - No transactions
  - General-purpose Indexes & Materialized Views newly available if using Cassandra 3
Cassandra for Online Systems

Keys for schema design:
1. Denormalize at write time for queries
2. Keep denormalized copies in sync at edit time
3. Avoid schemas that cause many, frequent edits on the same record
4. Avoid schemas that cause edit contention
5. Avoid inconsistency from read-before-write
Cassandra for Online Systems

What we did that worked:
1. Event sourced schema with multiple views
2. Event denormalization, with consistency checks
3. Flexible schema (JSON in blob)
4. Limits and throttling to deal with hotspots

- Details follow for Family Tree
Family Tree - Introduction

- Family Tree for the entire human family
  - 1.2B persons
  - 800M relationships
  - 7.8M registered users
  - 3.8M Family Tree contributors
- Free registration, Open Edit
- Supported by growing record collection
- World-wide user base
- Backed by Apache Cassandra (DSE)
Family Tree - Introduction

- Multiple views of person
  - Pedigree page
  - Person page
  - Person card popup
  - Person change history
  - Descendancy page
Pedigree Page

- 33 persons (plus children)
- 33 relationships (w/ details)
- 1 page view
Person Page (bottom)

- 18 persons (w/ details)
- 18 relationships (w/ details)
- 1 page view
Person Page (bottom)
James Mead Sumson

- BIRTH: 27 April 1873
  Springville, Utah, Utah, United States
- DEATH: 29 June 1958
  Springville, Utah, Utah, United States

22 SOURCES
6 DISCUSSIONS
7 MEMORIES

TEMPLE: B C I E SP SS
<table>
<thead>
<tr>
<th>Information</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attached</td>
<td>by Cherise Armstrong</td>
</tr>
<tr>
<td>Child Added</td>
<td>James Coplan Sumson 1928-2002</td>
</tr>
<tr>
<td>8 May 2017</td>
<td>by Cherise Armstrong</td>
</tr>
<tr>
<td>8 May 2017</td>
<td>by Cherise Armstrong</td>
</tr>
<tr>
<td>Source Attached</td>
<td>James M. Sumson, &quot;Utah Marriages, 1887-1935&quot;</td>
</tr>
<tr>
<td>24 March 2017</td>
<td>by tennynferney1</td>
</tr>
<tr>
<td>Source Attached</td>
<td>James Mead Sumson, &quot;Utah Marriages, 1887-1935&quot;</td>
</tr>
<tr>
<td>3 August 2016</td>
<td>by heathercurtis</td>
</tr>
<tr>
<td>Source Attached</td>
<td>James Mead Sumson, &quot;Utah Marriages, 1887-1935&quot;</td>
</tr>
<tr>
<td>26 June 2016</td>
<td>by ChristensenClydeS1</td>
</tr>
<tr>
<td>Source Attached</td>
<td>James Mead Sumson, &quot;Utah Marriages, 1887-1935&quot;</td>
</tr>
<tr>
<td>26 June 2016</td>
<td>by ChristensenClydeS1</td>
</tr>
<tr>
<td>Couple Event Added</td>
<td>Couple Event Marriage 04 Mar 1908 Salt Lake, Utah, United States</td>
</tr>
<tr>
<td>26 June 2016</td>
<td>by ChristensenClydeS1</td>
</tr>
<tr>
<td>Reference</td>
<td></td>
</tr>
</tbody>
</table>
Family Tree

- Flexible schema
- 4\textsuperscript{th} major iteration over 10 years
- Schema still adjusted relatively often (6 mo)
Family Tree

- **API stats:**
  - 300M API requests / peak day
  - 300K API requests / peak minute
  - 150M API requests / off-peak day

- **DB stats:**
  - 1.5B reads / peak day
  - 58K reads / sec (peak)
  - 10M writes / peak day
DB stats:
- 20TB of data (without 3x replication)
- 7.5TB of that is canonical
- 12.5TB is derivative, denormalized for queries

DB size:
- 60TB of disk used (replication factor = 3)
- Able to drop most derivative data in emergency
Family Tree

- **API performance**
  - Peak day P90 is 22ms (instead of 2-5 sec on Oracle)

- **DB performance**
  - Peak day P90 is 2.3ms
  - Peak day P99 is 9.9ms

- **Person page**
  - Able to be served from 2 person reads
  - Still lots of room for optimization
  - Front-end client still over-reading
Events & Views

- Events are CANONICAL
- Multiple, derivative views
  - View computed from events
  - Views can be deleted (recomputed from events)
- Views stored in DB
  - For faster reads
- Event Sourcing

https://martinfowler.com/eaaDev/EventSourcing.html
Events & Views

- Views optimized for Read
  - 100 reads : 1 write
- Different use case?
  - Might justify a new view
  - Might just change views
- Family Tree views
  - Person Card (summary)
  - Full Person View
  - Change History
Events & Views

- Types of reads
  - Full View Refresh
  - Incremental View Refresh
  - Fast Path Read (no refresh needed)
Events & Views

- Types of reads
  - Full View Refresh
  - Incremental View Refresh
  - Fast Path Read (no refresh needed)
Events & Views

- Types of reads
  - Full View Refresh
  - Incremental View Refresh
  - Fast Path Read (no refresh needed)
Read Optimizations
- Row Cache for view tables 14G (out of 60G)
- CL=ONE for Fast Path Read
- Upgrade to LOCAL_QUORUM
  - if read fails
  - if view refresh is required

Write Optimization
- Group events into tx record
- Split txs to avoid over-copy
Sample Cassandra Schema (event table):

```sql
CREATE TABLE person_journal (
    entity_id text,
    record_id timeuuid,
    sub_id int,
    type text,
    subtype text,
    content blob,
    primary key ((entity_id), record_id, sub_id, type, subtype))
with compaction = { 'class': 'SizeTieredCompactionStrategy' };
```
Sample Cassandra Schema (view table):

```sql
CREATE TABLE person_view (
    entity_id text,
    record_id timeuuid,
    sub_id int,
    type text,
    subtype text,
    content blob,
    primary key ((entity_id), record_id, sub_id, type, subtype))
    with caching = 'ALL'
and compaction = {
    'class': 'LeveledCompactionStrategy'
}
and gc_grace_seconds = 86400;
```
Classes of Writes:

1. Single record edits
2. Multiple record edits
   - 2-4 records
   - Simple changes
3. Composite multi-record edits
   - Many records
   - Complex changes
Consistency Model

Write Process:
1. Create & write single “command” record
2. Pre-read affected records (views)
3. Pre-apply events (non-durable)
4. Check for rule violations
5. Write events
6. Post-read new affected records
7. Check for rule violations
   ➢ Revert if problems
Failure Modes:

1. Rule violation
   - Bad request response
   - NO write

2. Race condition
   - Conflict response
   - Revert
Failure Modes:

3. Read Timeout at CL=ONE
   - Retry with LOCAL_QUORUM
   - Down node often is ignored

4. Write Timeout
   - Internal error response
   - Janitor follow-up later (from queue)
   - Idempotent writes
Surprises & Solutions

Surprises:
- Disproportionate Rate issues
- NTP Time issues
- Consistency issues
Disproportionate Rate

- Surprise: Bytes matter, not queries
  - Number of queries has less to do with latency
  - Large number of bytes cause CPU from Java GC
  - Multiple copies of large edited blobs add up
Disproportionate Rate

- Surprise: VERY Large Views
  - Well-known historical persons
  - Vanity genealogy (connecting to royalty)
  - 50+ names, 100+ spouses, 500+ parents
  - Many more bytes / request than normal (skews GC)
Disproportionate Rate

- Surprise: Single nodes matter, not total cluster
  - Slow node affects all traffic on that node
  - Events & Views on same node, worse hotspots

- Surprise: Replica set surprisingly resilient
Solution #1:
- Reduce size of views
- Family Tree data limits (control) & data cleanup (fix)
- Emergency blacklist for certain records until they can be manually trimmed

Solution #2:
- Throttle duplicate requests
- Throttle problem clients
- Reduce rate of requests to specific replica set
Solution #3:

- Spread views by prepending key prefix
- Events on different set of nodes than views
- Put each type of view on different set of nodes
- Spread traffic out

Solution #4:

- Prevent merge / edit wars (limits)
- Emergency lock records / suspend accounts
Disproportionate Rate

- **Solution #5:**
  - Split command up into contiguous events
  - Avoid over-copying large transactions
  - Split batches when writing
  - Retry writes if writes time out (janitor & queue)

- **Solution #6:**
  - Change view tables to LCS (leveled compaction)
  - Lower gc_grace_seconds for view tables to 2d
  - Emergency: Truncate view tables
Disproportionate Rate

- Solution #7:
  - Pre-compute common views
  - Spread out pub-sub consumers with queue delays
  - Prevents incremental view refresh races from pub-sub consumers
NTP Time

- NTP Time Issues:
  - Event transaction id is V1 time-based UUID
  - UUID generated on app server
  - Sequence of writes across app servers
  - App server time out of sync (broken NTP)
  - Arbitrary event reordering
Solution #1:
- Fix NTP config, of course
- Monitor / alert on NTP sync issues
Solution #2:
- Keep V1 UUIDs in sequence at write time
- Read prior UUID and wait up to 500ms until in past

This is the variation when fixed!
View Inconsistency

- Concurrent writes:
  - Concurrent incremental view refresh
  - Different view snapshots read (different nodes)
  - Overlapping view writes
  - Missing view data (as if write never happened)

- Partial writes:
  - Timeout on complex many-record write
  - Janitor not yet caught up replaying write
  - User refreshes and attempts again
View Inconsistency

- **Solution #1:**
  - Observe view UUID during event preparation
  - Observe view UUID during write
  - Revert if different (concurrent write conflict)

- **Solution #2:**
  - Spark job to find inconsistencies
  - Semi-automated categorization & fixup
  - Address each source of inconsistency
End Result

- Fantastic peak day performance
- Data consistency is good enough
- Consistency checks catching issues
- Quality of Family Tree improved with cleanups
- Splitting events / view – lots of flexibility
- Flexible schema – allows for agility
- Takes abuse from users and keeps running
End Result
End Result

18 months, incl. 8 months before cutover
Learnings

- Event Sourced data model:
  - Very performant & scalable
  - Good enough consistency

- NTP time:
  - Must monitor / alert
  - Must deal with small offsets

- Consistency checks:
  - Long-term consistency must be measured
  - Fixes for measured issues must be applied
Questions

- Thanks:
  - To Apache for hosting the conference
  - To all Cassandra contributors
  - To Datastax for DSE
  - To FamilySearch for sending me