Anti-Entropy using CRDTs on HA Datastores

Sailesh Mukil
Senior Software Engineer, Netflix
Timeline

- Cassandra adoption: 2011
- Multi-region: 2013
- Dynomite: 2016
Dynomite

Makes non-distributed datastores, distributed
Dynomite Overview

Datastore

33% 33% 33%
Dynomite Overview

Replica 1

Replica 2

Replica 3
Client

Replica 1

Replica 2

Replica 3
Dynomite overview

- Global replication
- High availability
- Shared nothing
- Auto-sharding
- Linear scale
- Pluggable datastores (Redis primarily)
- Multiple quorum levels
- Supports datastore API
Dynomite footprint @ Netflix

- ~1000 customer facing nodes
- ~1M OPS/s
- Largest cluster holds ~6 TB
The problem

Entropy in the system
Entropy in the system

R-1

SET K 123

R-2

R-3
Entropy in the system

SET K 123
Entropy in the system

R-1: K: 123

R-2: K: 123

R-3: K: 123

OK
Entropy in the system

K: 123

SET K 456
Entropy in the system

R-1
K: 123

R-2
K: 123

R-3
K: 456

SET K 456
Entropy in the system

K: 123

R-1

ERR

K: 123

R-2

K: 456

R-3
Entropy in the system

SET K 789

R-1

K: 123

R-2

K: 789

R-3

K: 456
Entropy in the system

K: 123

K: 789

K: 456

ERR
GET K (w/quorum)

K: 123

K: 789

K: 456
GET K (w/quorum)

K: 123

R-1

123

K: 789

R-2

K: 456

R-3
ERR:
QUORUM
FAILED

K: 123
R-1

K: 789
R-2

K: 456
R-3
Now if you'll excuse me, I need to go take a shower so I can't tell if I'm crying or not.
Replicas **will** go out of sync
Timeline

Cassandra adoption 2011
Multi-region 2013
Dynomite 2016
Dynamite w/ CRDTs 2019
Achieving anti-entropy
(traditionally)

Last Writer Wins
- Uses Physical timestamps
- Clock skew

Vector Clocks
- Shows causal relationships
- But not for concurrent writes
The solution

Conflict free replicated data types
Conflict free replicated data types

A CRDT is a data structure which can be replicated across the network, where the replicas can be updated independently and concurrently without coordination between the replicas, and where it is always mathematically possible to resolve inconsistencies which might result.
**Associative**

Grouping of operations does not matter

\[(X + Y) + Z = X + (Y + Z)\]

**Commutative**

Order of operations do not matter

\[X + Y = Y + X\]

**Idempotent**

Duplication of operations does not matter

\[X + X = X\]
Types of operations on CRDTs

Update
- Updates local state

Merge
- Converges replica states
When we write, we **update**

When we repair, we **merge**

Read repair = **merge** on read path
CRDTs provide strong eventual consistency
Naive distributed counter

CTR: 1

INCR CTR

R-1

CTR: 1

R-2

CTR: 1

R-3
Naive distributed counter

CTR: 1

INCR CTR

CTR: 2

DECR CTR

CTR: 0
Naive distributed counter

CTR: 1

Repair based on timestamp?

Latest value is 2, which is incorrect

CTR: 0

CTR: 2
CRDT: PNCounters

Each replica maintains 2 “local” counters

- Positive counter: Tracks increments
- Negative counter: Tracks decrements

Final counter value:

(Sum of all PCounters - Sum of all NCounters)
CRDT: PNCounter

INCR CTR

CTR: 1 0 0
0 0 0
0 0 0

CTR: 1 0 0
0 0 0
0 0 0

CTR: 1 0 0
0 0 0
0 0 0

CTR: 1 0 0
0 0 0
0 0 0
CRDT: PNCounter

CTR:

1 0 0
0 0 0

DECR CTR

INCR CTR

R-1

R-2

R-3
CRDT: PNCounter

CTR = 0
CTR = 1
CTR = 2
CRDT: PNCounter

R-1

CTR:
0
0
0
0

R-2

CTR:
1
0
0
0

R-3

CTR:
1
0
1
1

GET CTR
CRDT: PNCounter

```
CTR:
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
GET CTR
```

```
CTR:
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
repair (merge)
```

```
CTR:
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
CRDT: PNCounter

CTR:

1 0 1
0 1 0
CTR = 1

CTR:

1 0 1
0 1 0
CTR = 1

CTR:

1 0 1
0 1 0
CTR = 1
CRDT: LWW-Element Set

Used for registers, hashmaps and sorted sets

Used to maintain key metadata
- Add set: Latest update timestamps for keys
- Remove set: Timestamps at which keys were removed

Registers can take arbitrary values
- Hence we still require LWW to resolve conflicts
LWW-Element
Set

R-1

add
K1
rem

add
rem

K1: 123

R-2

add
K1
rem

add
rem

K1: 123

R-3

add
K1
rem

add
rem

K1: 456

SET K1 456 (t2)
LWW-Element

Set

**SET K2 999 (t3)**

R-1

<table>
<thead>
<tr>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>t3</td>
</tr>
</tbody>
</table>

K1: 123  K2: 999

R-2

<table>
<thead>
<tr>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>t3</td>
</tr>
</tbody>
</table>

K1: 123  K2: 999

R-3

<table>
<thead>
<tr>
<th>K1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
</tr>
</tbody>
</table>

K1: 456

---

NETFLIX
LWW-Element Set

add
K1  K2
K1: t2  t3
rem
K1:  

K1 = 123 (t1)

R-1

add
K1  K2
K1: t1  
rem
K1:  

K1 = 456 (t2)

R-2

add
K1  K2
K1: t1  
rem
K1:  

R-3

add
K1  
rem
K1:  

GET K1

K1 = 456 (t2)

K1 = 123 (t1)

t2 > t1

=> 456 latest value

repair
LWW-Element Set

R-1

add
K1 K2
rem
K1: 456 K2: 999

R-2

add
K1 K2
t2 t3
rem
K1: 456 K2: 999

R-3

add
K1
t2
rem
K1: 456

“456”

repair

K1: 123
K2: 999

K1: 456
K2: 999
GET K2

K2 = 999 (t3)
LWW-Element Set

R-1

```
<table>
<thead>
<tr>
<th>add</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rem</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
</table>

K1: 456  K2: 999
```

R-2

```
<table>
<thead>
<tr>
<th>add</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rem</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
</table>

K1: 456  K2: 999
```

R-3

```
<table>
<thead>
<tr>
<th>add</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rem</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
</table>

K1: 456  K2: 999
```

Repair

```
<table>
<thead>
<tr>
<th>K2</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>
```

R-1

```
<table>
<thead>
<tr>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>456</td>
<td></td>
</tr>
</tbody>
</table>
```

R-2

```
<table>
<thead>
<tr>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>456</td>
<td></td>
</tr>
</tbody>
</table>
```

R-3

```
<table>
<thead>
<tr>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td>456</td>
<td></td>
</tr>
</tbody>
</table>
```
LWW-Element
Set

DEL K2 (t4)

K1 | K2
---|---
456 | 999

add
rem

K1: 456  K2: 999

R-2

K1 | K2
---|---
456 | 999

t2 | t3

add
rem

K1: 456  K2: 999

R-1

K1: 456  K2: 999

R-3

K1 | K2
---|---
456 | 999

t2 | t3

add
rem
### LWW-Element Set

**Add Operations**

- **R-1**
  - Add: K1: t2, K2: t3

**Remove Operations**

- **R-2**
  - Add: K1: t2
  - Remove: K2: t4

- **R-3**
  - Add: K1: t2, K2: t3

**Get K2**

- Value: "999"

**Data Structure**

<table>
<thead>
<tr>
<th></th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Add</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t2</td>
<td>t3</td>
</tr>
<tr>
<td><strong>Remove</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t2</td>
<td>t3</td>
</tr>
</tbody>
</table>

**Key Values**

- K1: 456
- K2: 999
LWW-Element Set

R-2

<table>
<thead>
<tr>
<th>add</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t2</td>
<td>t3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rem</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

R-3

<table>
<thead>
<tr>
<th>add</th>
<th>K1</th>
<th>K2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t2</td>
<td>t4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rem</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

K2: 999
R-1

K2 = 999 (t3)

K2 del @t4

GET K2

K2: 999

R-1

K2 = 999 (t3)
Implementation challenges (LWW-element set)

Redis doesn’t maintain timestamps.

Dynomite can track the timestamp of the client request.
Implementation challenges (LWW-element set)

We’d like Dynomite to remain stateless

Store the metadata inside Redis
Implementation challenges (LWW-element set)

Operations must modify data and metadata atomically

Rewrite operations into Redis Lua scripts (guarantees atomicity)
Implementation challenges (LWW-element set)

Does the remove set grow forever?

Delete metadata ASAP from remove set if ALL replicas agree
Background thread cleans rest
Maintain remove set as sorted set
Implementation challenges (LWW-element set)

What does an example Lua script look like?

Check if update is old
Discard if it is
Update data + metadata otherwise
Repairs occur on read path in Dynomite

Repairs for point reads only
Background repairs

(Note: Ongoing work)
Repairing on range reads is expensive

Eg: Give me all members of a set
Return everything in this hashmap
Return me a range from this sorted set
How do we target keys that need repairing?

Full key walk? (like Cassandra)
How do we target keys that need repairing?

Maintain list of recently written to keys
Run merge operation on them (async)
But, merge operation on large structures are expensive
Background repairs

Delta-state CRDTs

Maintain list of recent mutations done to keys
Ship only delta-state instead of entire data structure for merge
Confirm which replicas have received it
Background repairs

INCR CTR

What is a delta-state?

CTR:

R1

2 0
0 1

Full state

CTR:

R2

2 0
0 1

1 0
0 1
What is a delta-state?

Background repairs

INCR CTR

Delta state

CTR: 2 0
0 1

R1 = 2

CTR: 2 0
0 1

R2
What is a delta-state?

Background repairs

<table>
<thead>
<tr>
<th>Mutations</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta-1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\delta-2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\delta-3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\delta-4$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ACK

ACK

R1

R2

R3
What is a delta-state?

<table>
<thead>
<tr>
<th>Mutations</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta^{-1}$</td>
<td>ack</td>
<td>ack</td>
</tr>
<tr>
<td>$\delta^{-2}$</td>
<td>ack</td>
<td>ack</td>
</tr>
<tr>
<td>$\delta^{-3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\delta^{-4}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## What is a delta-state?

<table>
<thead>
<tr>
<th>Mutations</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>δ-1</td>
<td>ack</td>
<td>ack</td>
</tr>
<tr>
<td>δ-2</td>
<td>ack</td>
<td>ack</td>
</tr>
<tr>
<td>δ-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>δ-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ACK
What is a delta-state?

Background repairs

<table>
<thead>
<tr>
<th>Mutations</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>δ-1</td>
<td>aek</td>
<td>aek</td>
</tr>
<tr>
<td>δ-2</td>
<td>aek</td>
<td>aek</td>
</tr>
<tr>
<td>δ-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>δ-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Challenge with Delta-state CRDTs

Durability

Practical overhead of maintaining list
Thank You.

Sailesh Mukil
smukil@netflix