Claret
Using Data Types for High Contention Distributed Transactions

Brandon Holt, Irene Zhang, Dan Ports, Mark Oskin, Luis Ceze
UNIVERSITY of WASHINGTON
PaPoC’15 @ EuroSys
Brandon Holt @holtbg
At #EuroSys right now!

EuroSys 2015 @EuroSys2015
Co-located workshop: Principles and Practice of Consistency for Distributed Data.
papoc.di.uminho.pt

Ellen DeGeneres @TheEllenShow
If only Bradley's arm was longer. Best photo ever. #oscars
At #EuroSys right now!

Co-located workshop: Principles and Practice of Consistency for Distributed Data. papoc.di.uminho.pt

If only Bradley’s arm was longer. Best photo ever. #oscars
Brandon Holt @holtbg
At #EuroSys right now!

Ellen DeGeneres @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

post[1003] ⇒ Post
  content: “If only Bradley’s arm was longer. Best photo ever. #oscars”

retweets[1003] ⇒ Set
  user:43  user:10  user:29  user:74  user:89
Brandon Holt @holtbg
At #EuroSys right now!

Ellen DeGeneres @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

post[1003] $\Rightarrow$ Post
{ 
  author: user:92
  content: “If only Bradley’s arm was longer. Best photo ever. #oscars”
}

retweets[1003] $\Rightarrow$ Set
{ 
  user:43
  user:10
  user:29
  user:89
  user:74
}

Retweet
retweets[1003].add("user:53")
Brandon Holt @holtbg
At #EuroSys right now!

Ellen DeGeneres @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

post[1003] ⇒ Post
{author: user:92
  content: "If only Bradley’s arm was longer. Best photo ever. #oscars"
}

retweets[1003] ⇒ Set
{user:43 user:10
  user:29
  user:74
  user:89
}

Retweet
retweets[1003].add("user:53")

View post
retweet_count = retweets[1003].size()
# ...
How do we make this scale?

```python
post[1003] ⇒ Post
  author: user:92
  content: “If only Bradley’s arm was longer. Best photo ever. #oscars”

retweets[1003] ⇒ Set
  user:43
  user:10
  user:29
  user:74

Retweet
retweets[1003].add("user:53")

View post
retweet_count = retweets[1003].size()
# ...
```
NoSQL
**Ellen DeGeneres** @TheEllenShow

If only Bradley’s arm was longer. Best photo ever. #oscars

**Brandon Holt** @holtbg

At #EuroSys right now!

---

**EuroSys 2015** @EuroSys2015

Co-located workshop: Principles and Practice of Consistency for Distributed Data. papoc.di.uminho.pt

---

NoSQL

```
post:1003:author  =>  92
post:1003:content =>  "If only Bradley’s arm was longer. Best photo ever. #oscars"
retweeters:1003   =>  "user29,user:89,user:74,
                      user:10,user:43"
```

```
Retweet
s = get("retweeters:1003")
if "user:43" not not in s:
    s += "user:43"
put("retweeters:1003", s)
```

```
View post
retweets = get("retweeters:1003")
# ...
```

must be atomic

which retweets will this contain?
Transactions?

"Too expensive."

"Don’t scale."

What if the datastore knew more?

More information → more chance for optimization

Opportunity:

Use data types provided by the programmer

transactions? which retweets will this contain?

View post

```
retweets = get("retweets:1003")
# ...
```
Abstract Data Types in NoSQL
- programmers express *intent* through types
- *flexible* data model, no fixed schema
- leverage *ADT properties* for transaction performance
- *sanely* trade off consistency for scalability
Leveraging Abstract Data Types in NoSQL

Commutativity
- Transactional boosting
- Combining

Approximate data types
- Bounded inconsistency
- Isolated eventual consistency (CRDTs)
- Probabilistic data types

Evaluation: Claret prototype
Leveraging Abstract Data Types in NoSQL

Commutativity
- Transactional boosting
- Combining

Approximate data types
- Bounded inconsistency
- Isolated eventual consistency (CRDTs)
- Probabilistic data types

Evaluation:
Commutativity

Brandon Holt @holtbg
At #EuroSys right now!

EuroSys 2015 @EuroSys2015
Co-located workshop: Principles and Practice of Consistency for Distributed Data.
papoc.di.uminho.pt

Ellen DeGeneres @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

View post
post = Map("post:1003").get()
retweets = Set("retweeters:1003").size()
# ...

post:1003 ⇒
  author: 92
  content: "If only Bradley’s arm was longer. Best photo ever. #oscars"

retweeters:1003 ⇒
  user:43
  user:10
  user:29
  user:89
  user:74

many reads → okay
Commutativity

Brandon Holt  @holtbg
At #EuroSys right now!

EuroSys 2015  @EuroSys2015
Co-located workshop: Principles and Practice of Consistency for Distributed Data.
papoc.di.uminho.pt

Ellen DeGeneres  @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

post:1003 ➞
author: 92
content: "If only Bradley’s arm was longer. Best photo ever. #oscars"

user:43  user:10
user:29  user:89  user:74

Retweet
Set("retweeters:1003").add("user:53")

Retweet
Set("retweeters:1003").add("user:53")
Commutativity

Brandon Holt @holtbg
At #EuroSys right now!

EuroSys 2015 @EuroSys2015
Co-located workshop: Principles and Practice of Consistency for Distributed Data.
papoc.di.uminho.pt

Ellen DeGeneres @TheEllenShow
If only Bradley's arm was longer. Best photo ever. #oscars

retweeters:1003 \implies
\{author: 92, content: "If only Bradley’s arm was longer. Best photo ever. #oscars"\}

Retweet
Set("retweeters:1003").add("user:53")

Retweet
Set("retweeters:1003").add("user:53")
Commutativity

Brandon Holt @holtbg
At #EuroSys right now!

EuroSys 2015 @EuroSys2015
Co-located workshop: Principles and Practice of Consistency for Distributed Data.
papoc.di.uminho.pt

Ellen DeGeneres @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

many updates → contention

post:1003
author: 92
content: "If only Bradley’s arm was longer. Best photo ever. #oscars"

retweeters:1003
user:43
user:10
user:29
user:89
user:74

Retweet
Set("retweeters:1003").add("user:53")
Commutativity

Brandon Holt  @holtbg
At #EuroSys right now!

EuroSys 2015  @EuroSys2015
Co-located workshop: Principles and Practice of Consistency for Distributed Data.
papoc.di.uminho.pt

Ellen DeGeneres  @TheEllenShow
If only Bradley’s arm was longer. Best photo ever. #oscars

post:1003  ➞
author: 92
content: "If only Bradley’s arm was longer. Best photo ever. #oscars"

retweeters:1003  ➞
user:43  user:10
user:29  user:89  user:74

Set adds commute!

Retweet
Set("retweeters:1003").add("user:53")
# add post to followers’ timelines
Commutativity

For a given data type: which pairs of operations commute?

Commutativity Specification* for Set

<table>
<thead>
<tr>
<th>method</th>
<th>commutes with:</th>
<th>when:</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(x): void</td>
<td>add(y)</td>
<td>∀x,y</td>
</tr>
<tr>
<td>remove(x): void</td>
<td>remove(y)</td>
<td>∀x,y</td>
</tr>
<tr>
<td></td>
<td>add(y)</td>
<td>x ≠ y</td>
</tr>
<tr>
<td>size(): int</td>
<td>add(x)</td>
<td>x ∈ Set</td>
</tr>
<tr>
<td></td>
<td>remove(x)</td>
<td>x ∉ Set</td>
</tr>
<tr>
<td>contains(x): bo</td>
<td>add(y)</td>
<td>x ≠ y ∨ y ∈ Set</td>
</tr>
<tr>
<td></td>
<td>remove(y)</td>
<td>x ≠ y ∨ y ∉ Set</td>
</tr>
<tr>
<td></td>
<td>size()</td>
<td>∀x</td>
</tr>
</tbody>
</table>

**Commutativity**

For a given data type: which pairs of operations commute?

---

### Commutativity Specification* for Set

<table>
<thead>
<tr>
<th>method:</th>
<th>commutes with:</th>
<th>when:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(x): void</code></td>
<td><code>add(y)</code></td>
<td><code>∀x,y</code></td>
</tr>
<tr>
<td><code>remove(x): void</code></td>
<td><code>remove(y)</code></td>
<td><code>∀x,y</code></td>
</tr>
<tr>
<td><code>size(): int</code></td>
<td><code>add(x)</code></td>
<td><code>x ∈ Set</code></td>
</tr>
<tr>
<td></td>
<td><code>remove(x)</code></td>
<td><code>x ∉ Set</code></td>
</tr>
<tr>
<td><code>contains(x): bool</code></td>
<td><code>add(y)</code></td>
<td><code>x ≠ y ∨ y ∈ Set</code></td>
</tr>
<tr>
<td></td>
<td><code>remove(y)</code></td>
<td><code>x ≠ y ∨ y ∉ Set</code></td>
</tr>
<tr>
<td></td>
<td><code>size()</code></td>
<td><code>∀x</code></td>
</tr>
</tbody>
</table>


If the key/value store knew this, what could it do?
**Problem:** contention $\rightarrow$ many aborts / retries

**Commutativity**

T1
```plaintext
Set("retweeters:1003").add(53)
# add post to followers' timelines
f = followers(53).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

T2
```plaintext
Set("retweeters:1003").add(89)
# add post to followers' timelines
f = followers(89).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```
Commutativity

**Problem:** contention $\rightarrow$ many aborts / retries

```
T1
Set("retweeters:1003").add(53)
# add post to followers' timelines
f = followers(53).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

```
T2
Set("retweeters:1003").add(89)
# add post to followers' timelines
f = followers(89).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```
**Commutativity**

**Problem:** contention $\rightarrow$ many aborts / retries

```
T1
Set("retweeters:1003").add(53)
# add post to followers' timelines
f = followers(53).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

```
T2
Set("retweeters:1003").add(89)
# add post to followers' timelines
f = followers(89).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```
Commutativity

Problem:

Solution: *Transactional boosting*
- when operations commute, no need to abort their transactions

```
T1
Set("retweeters:1003").add(53)
# add post to followers’ timelines
f = followers(53).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)

T2
Set("retweeters:1003").add(89)
# add post to followers’ timelines
f = followers(89).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

---

* M. Herlihy and E. Koskinen.
**Commutativity**

**Problem:**

**Solution:** *Transactional boosting*
- when operations commute, no need to abort their transactions
- reduce abort rate → increase throughput

```
T1
Set("retweeters:1003").add(53)
# add post to followers' timelines
f = followers(53).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

```
T2
Set("retweeters:1003").add(89)
# add post to followers' timelines
f = followers(89).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

```
T3
Set("retweeters:1003").add(71)
# add post to followers' timelines
f = followers(71).all()
timeline(f[0]).push(1003)
timeline(f[1]).push(1003)
timeline(f[2]).push(1003)
timeline(f[3]).push(1003)
```

Commutativity

Problem: Serializing operations on hot records

Set("retweeters:1003").add(53)
Set("retweeters:1003").add(89)
Set("retweeters:1003").add(71)
Set("retweeters:1003").add(22)
Set("retweeters:1003").add(11)
Set("retweeters:1003").add(55)
Set("retweeters:1003").add(42)
Set("retweeters:1003").add(91)
Set("retweeters:1003").add(96)
Commutativity

Problem:

Set("retweeters:1003").add(53)
Set("retweeters:1003").add(89)
Set("retweeters:1003").add(71)
Set("retweeters:1003").add(22)
Set("retweeters:1003").add(11)
Set("retweeters:1003").add(55)
Set("retweeters:1003").add(42)
Set("retweeters:1003").add(91)
Set("retweeters:1003").add(96)

* D. Hendler, I. Incze, N. Shavit, and M. Tzafrir.
Commutativity

Problem:

Solution: *Combining*
- merge multiple operations together and apply them all at once

\[
\text{Set("retweeters:1003").add(53)}
\]
\[
\text{Set("retweeters:1003").add(89)} \rightarrow \text{Set("retweeters:1003").add([53, 89, 71])}
\]
\[
\text{Set("retweeters:1003").add(71)}
\]
\[
\text{Set("retweeters:1003").add(22)}
\]
\[
\text{Set("retweeters:1003").add(11)} \rightarrow \text{Set("retweeters:1003").add([22, 11, 55])}
\]
\[
\text{Set("retweeters:1003").add(55)}
\]
\[
\text{Set("retweeters:1003").add(42)}
\]
\[
\text{Set("retweeters:1003").add(91)} \rightarrow \text{Set("retweeters:1003").add([42, 91, 96])}
\]
\[
\text{Set("retweeters:1003").add(96)}
\]

* D. Hendler, I. Incze, N. Shavit, and M. Tzafrir.
Problem: Commutativity

Solution: **Combining**

- merge multiple operations together and apply them all at once
- parallelize and decrease contention

* D. Hendler, I. Incze, N. Shavit, and M. Tzafrir.
Leveraging **Abstract Data Types** in NoSQL

**Commutativity**
- Transactional boosting
- Combining

**Approximate data types**
- Bounded inconsistency
- Isolated eventual consistency (CRDTs)
- Probabilistic data types

**Evaluation:** *Claret* prototype
Approximate data types

**Problem:** Reads don’t commute with updates

```
Retweet
Set("retweeters:1003").add("user:53")
# ...
```

```
View post
# ...
retweets = Set("retweeters:1003").size()
# ...
```
Approximate data types

**Problem:** Reads don’t commute with updates

```python
Retweet
Set("retweeters:1003").add("user:53")
# ...
```

```python
View post
# ...
retweets = Set("retweeters:1003").size()
# ...
```

doesn’t need to be precise
Approximate data types

Problem:

Solution: *Bounded inconsistency*

- allow *some* updates concurrently with reads
- exposes additional "commutativity"

Solution:

```python
Retweet
Set("retweeters:1003").add("user:53")
# ...
```

View post

```python
retweets = Set("retweeters:1003").approxSize<0.05>()
# ...
```
Approximate data types

Problem:

Solution: *Bounded inconsistency*
- allow *some* updates concurrently with reads
- exposes additional "commutativity"

Retweet

```
Set("retweeters:1003").add("user:53")
# ...```

View post

```
retweets = Set("retweeters:1003").approxSize<0.05>()
# ...```

5% error → 170,000 adds
Approximate data types

**Problem:** Scaling $\rightarrow$ high latencies, low availability
Approximate data types

**Problem:** Scaling $\rightarrow$ high latencies, low availability

- Replica 0
- Replica 1
- Replica 2

everything replicated, all eventual consistency
Approximate data types

Problem:

**Solution:** Isolated eventual consistency via CRDTs
- use CRDT data type only where needed for scaling or low-latency
- programmers choose what can be approximate
Approximate data types

**Problem:** Can’t (or don’t want to) store all the data
Approximate data types

Problem:
Problem:

**Solution:** *Probabilistic data types*
- e.g. HyperLogLog, Bloom filter, Count-min sketch, T-digest
- useful for tracking statistics, summary of high-volume data, or partially-materialized views
Leveraging **Abstract Data Types in NoSQL**

**Commutativity**
- Transactional boosting
- Combining

**Approximate data types**
- Bounded inconsistency
- Isolated eventual consistency (CRDTs)
- Probabilistic data types

**Evaluation:** *Claret* prototype
- Transactional boosting
- Bounded inconsistency
Evaluation

**Claret**: Key-value store with data types

- simple two-phase commit protocol with locking
  (+*transactional boosting*)
- experiments run with 4 shards, standard local ethernet network, 8-core 2GHz Intel Xeon processor per node
Evaluation

Case study: Twitter clone

- realistic synthetic graph (Kronecker, scale 14)
- simple random user model, retweet more popular posts (viral effect)
Evaluation

Case study: Twitter clone

- realistic synthetic graph (Kronecker, scale 16)
- simple random user model, retweet more popular posts (viral effect)
Evaluation

Case study: Twitter clone
- realistic synthetic graph (Kronecker, scale 16)
- simple random user model, retweet more popular posts (viral effect)
Evaluation

Case study: Twitter clone

- realistic synthetic graph (Kronecker, scale 16)
- simple random user model, retweet more popular posts (viral effect)
Claret

Abstract Data Types for NoSQL
Flexible data model lets programmers express intent

Commutativity
Leverage type info for transaction performance

Approximate data types
Sanely trade off consistency for scalability