Strong and Efficient Consistency with Consistency-Aware Durability

Aishwarya Ganesan, Ram Alagappan, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau
Distributed Storage Systems

- Twitter
- Facebook
- Lyft
- Airbnb
- Amazon
- MongoDB
- Kafka
- Cassandra
- Redis
- Apache ZooKeeper™
Consistency Models in Distributed Systems
Consistency Models in Distributed Systems

What does a read see given a previous set of reads and writes?
Consistency Models in Distributed Systems

What does a read see given a previous set of reads and writes?

**strong • linearizability**
Consistency Models in Distributed Systems

What does a read see given a previous set of reads and writes?

- strong
- linearizability
- weak
- eventual
Consistency Models in Distributed Systems

What does a read see given a previous set of reads and writes?

![Consistency Models Diagram]
Consistency Models in Distributed Systems

What does a read see given a previous set of reads and writes?

Well studied and understood!
Durability Models

Unlike consistency models, scant attention to durability model!
Durability Models

Unlike consistency models, scant attention to durability model!

How writes are replicated and persisted
Durability Models

Unlike consistency models, scant attention to durability model!

How writes are replicated and persisted

Durability model influences consistency
Durability Models

Unlike consistency models, scant attention to durability model!

How writes are replicated and persisted

Durability model influences consistency

Also determines performance
Durability Models

Unlike consistency models, scant attention to durability model!

How writes are replicated and persisted

Durability model influences consistency

Also determines performance

Despite this importance, often overlooked!
Two Widely Used Durability Models
Two Widely Used Durability Models

Immediate durability
Two Widely Used Durability Models

Immediate durability

write
client

node-1 node-2 node-3
Two Widely Used Durability Models

Immediate durability

client

replicate

write

node-1
node-2
node-3
Two Widely Used Durability Models

Immediate durability

write

client

replicate

persist

node-1

node-2

node-3
Two Widely Used Durability Models

Immediate durability

- Write to client
- Ack
- Replicate persist
- Fsync on node-1, node-2, node-3
Two Widely Used Durability Models

Immediate durability

enables strong consistency
but too slow!
Two Widely Used Durability Models

Immediate durability

- Client writes to node-1
- Node-1 sends an ack
- Replicate persist

Eventual durability

- Node-1, node-2, and node-3 perform fsync

Enables strong consistency but too slow!
Two Widely Used Durability Models

Immediate durability

- Client write
- Node-1 fsync
- Node-2 fsync
- Node-3 fsync
- Persist

Eventual durability

- Client write
- Node-1 fsync
- Node-2 fsync
- Node-3 fsync

enables strong consistency but too slow!
Two Widely Used Durability Models

Immediate durability

- Client write
- Node 1 fsync
- Node 2 fsync
- Node 3 fsync
- Replicate
- Persist

Enables strong consistency but too slow!

Eventual durability

- Client write
- Node 1 ack
- Lazily replicate
- Persist

Node 1
Node 2
Node 3
Two Widely Used Durability Models

Immediate durability
- Enables strong consistency but too slow!

Eventual durability
- Fast
- but enables only weak consistency due to data loss upon failures!

replicate persist

write

lazily replicate and persist

ack
Is it possible for a durability layer to enable both strong consistency and high performance?
CAD: Consistency-aware Durability
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account.
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account

Intuition: what a read sees is important for most consistency models
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account

Intuition: what a read sees is important for most consistency models

Key idea: CAD shifts the point of durability to reads from writes

  data is replicated and persisted before a read is served
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account

Intuition: what a read sees is important for most consistency models

Key idea: CAD shifts the point of durability to reads from writes
  - data is replicated and persisted before a read is served
  - delayed writes → high performance
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account

**Intuition:** what a read sees is important for most consistency models

**Key idea:** CAD shifts the point of durability to reads from writes
- data is replicated and persisted *before a read is served*
- delayed writes → high performance
- data durable before it is read → strong consistency even under failures
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account

**Intuition:** what a read sees is important for most consistency models

**Key idea:** CAD shifts the point of durability to reads from writes

- data is replicated and persisted before a read is served
- delayed writes → high performance
- data durable before it is read → strong consistency even under failures
- lose some writes if failures arise before read; but, useful for many systems that use eventual durability
CAD: Consistency-aware Durability

Design the durability layer by taking the consistency model into account

Intuition: what a read sees is important for most consistency models

Key idea: CAD shifts the point of durability to reads from writes
   data is replicated and persisted before a read is served
   delayed writes \(\rightarrow\) high performance
   data durable before it is read \(\rightarrow\) strong consistency even under failures
   lose some writes if failures arise before read; but, useful for many systems that use eventual durability

We show efficacy of CAD by providing cross-client monotonic reads
   a new and strong consistency property
Results
Results

ORCA: CAD and cross-client monotonic reads for leader-based systems implemented in ZooKeeper
Results

**ORCA**: CAD and cross-client monotonic reads for leader-based systems implemented in ZooKeeper

Compared to strongly consistent ZooKeeper

- ORCA is $1.6 - 3.3x$ faster by using CAD
- higher read throughput by allowing reads at many nodes
- reduces latency in geo-distributed settings by $14x$
Results

**ORCA**: CAD and cross-client monotonic reads for leader-based systems implemented in ZooKeeper

Compared to strongly consistent ZooKeeper
- ORCA is 1.6 – 3.3x faster by using CAD
- Higher read throughput by allowing reads at many nodes
- Reduces latency in geo-distributed settings by 14x

Compared to weakly consistent ZooKeeper
- ORCA provides similar throughput and latency
- But with stronger guarantees
Results

**ORCA:** CAD and cross-client monotonic reads for leader-based systems implemented in ZooKeeper

Compared to strongly consistent ZooKeeper

- ORCA is 1.6 – 3.3x faster by using CAD
- Higher read throughput by allowing reads at many nodes
- Reduces latency in geo-distributed settings by 14x

Compared to weakly consistent ZooKeeper

- ORCA provides similar throughput and latency
- But with stronger guarantees

Experimentally show ORCA’s guarantees under failures, useful for apps
Outline

Introduction
Motivation
CAD and cross-client monotonic reads
ORCA design
Results
Summary and conclusion
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability

latest data: no staleness
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability

latest data: no staleness
in-order reads across clients
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability

latest data: no staleness
in-order reads across clients

Weaker models
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

**Linearizability**

- latest data: no staleness
- in-order reads across clients

**Weaker models**
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

**Linearizability**

- **latest data:** no staleness
- **in-order reads across clients**

**Weaker models**
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

Linearizability

- latest data: no staleness
- in-order reads across clients

Weaker models

- stale reads
- out-of-order reads across clients
Consistency Models and Guarantees

Example: I’m bored at FAST and want to go home!

**Linearizability**

- **latest data:** no staleness
- in-order reads across clients

**Weaker models**

- **stale reads**
- **out-of-order** reads across clients
  even with monotonic reads and causal
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

leader – S1

S2

S3

in memory on disk
durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

leader – S1

S2

S3

in memory

on disk
durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

\[ \text{set } A = a_1 \]

leader – S1

S2

S3

in memory on disk durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

set $A = a_1$

leader – S1

S2

S3

in memory on disk

durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

```
set A = a1

leader - S1: a0 a1
S2: a0 a1
S3: a0 a1
```

- in memory
- on disk
- durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

![Diagram showing the process of setting a value and acknowledging it among replicas.]

- set $A=a_1$
- leader - $S1$
- $S2$
- $S3$

- in memory
- on disk

**durable** = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

![Diagram](image)

- in memory
- on disk
- durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

set $A = a_1$

leader – $S1$

$S2$

$S3$

in memory on disk durable = on disk on a majority
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures
Realizing Strong Consistency

Linearizability requires immediate durability
must synchronously replicate and persist data on a majority to tolerate failures

Poor performance due to synchronous operations
10x slower within data center
Weaker models only require eventual durability
  data buffered on one node, replication and persistence in background
Realizing Weaker Models

Weaker models only require eventual durability

data buffered on one node, replication and persistence in background

app

session-1

\[ \text{set } A = a \]

\[ \text{S1} \]
\[ a_0 \]

\[ \text{S2} \]
\[ a_0 \]

\[ \text{S3} \]
\[ a_0 \]
Realizing Weaker Models

Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Realizing Weaker Models

Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Realizing Weaker Models

Weaker models only require eventual durability
data buffered on one node, replication and persistence in background
Realizing Weaker Models

Weaker models only require eventual durability
data buffered on one node, replication and persistence in background

out-of-order across clients
valid under causal and monotonic reads but confusing semantics
Weaker models only require eventual durability
data buffered on one node, replication and persistence in background

Many deployments prefer eventual durability for performance
in fact, it is the default (e.g., MongoDB, Redis)
Realizing Weaker Models

Weaker models only require eventual durability
data buffered on one node, replication and persistence in background

Many deployments prefer eventual durability for performance
in fact, it is the default (e.g., MongoDB, Redis)
Thus settle for weak consistency
Immediate durability enables strong consistency but is slow.
Eventual durability is fast but enables only weaker consistency.

Many deployments prefer eventual durability for performance in fact, it is the default (e.g., MongoDB, Redis).
Thus settle for weak consistency.
Outline

Introduction
Motivation
CAD and cross-client monotonic reads
ORCA design
Results
Summary and conclusion
Consistency-aware Durability
Consistency-aware Durability

Most consistency models care about what reads see
Consistency-aware Durability

Most consistency models care about what reads see

Key idea: CAD shifts the point of durability to reads from writes
Consistency-aware Durability

Most consistency models care about what reads see
Key idea: CAD shifts the point of durability to reads from writes

delay durability of writes
Consistency-aware Durability

Most consistency models care about what reads see.

Key idea: CAD shifts the point of durability to reads from writes.

Delay durability of writes.
Consistency-aware Durability

Most consistency models care about what reads see

Key idea: CAD shifts the point of durability to reads from writes

delay durability of writes
Consistency-aware Durability

Most consistency models care about what reads see.
Key idea: CAD shifts the point of durability to reads from writes.

Delay durability of writes.
Consistency-aware Durability

Most consistency models care about what reads see

Key idea: CAD shifts the point of durability to reads from writes

delay durability of writes

write

client

S1

S2

S3

good performance
Consistency-aware Durability

Most consistency models care about what reads see

Key idea: CAD shifts the point of durability to reads from writes

- delay durability of writes
- make data durable before serving reads

Client

S1

S2

S3

write

ack

good performance

client

write

ack
Most consistency models care about what reads see

Key idea: CAD shifts the point of durability to reads from writes

Consistency-aware Durability

delay durability of writes

make data durable before serving reads

good performance
Consistency-aware Durability

Most consistency models care about what reads see
Key idea: CAD shifts the point of durability to reads from writes

delay durability of writes

make data durable before serving reads

good performance
Consistency-aware Durability

Most consistency models care about what reads see.

Key idea: **CAD shifts the point of durability** to reads from writes.

<table>
<thead>
<tr>
<th>Delay durability of writes</th>
<th>Make data durable before serving reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>write S1 → S2 → S3</td>
<td>read S1 → S2 → S3</td>
</tr>
</tbody>
</table>

**S1**: write to S1
**S2**: acknowledge S1
**S3**: make data durable

Good performance
Consistency-aware Durability

Most consistency models care about what reads see

Key idea: CAD shifts the point of durability to reads from writes

- delay durability of writes
- make data durable before serving reads

**Good performance**
Consistency-aware Durability

Most consistency models care about what reads see.

Key idea: CAD shifts the point of durability to reads from writes.

- **Delay durability of writes**
  - Good performance
  - Prevents out-of-order data across failures

- **Make data durable before serving reads**
  - Strong consistency
Most consistency models care about what reads see

Key idea: CAD **shifts the point of durability** to reads from writes

CAD does not always incur overheads on reads
reads do not immediately follow writes – natural in many workloads
common case: data already durable well before applications access it

---

**Consistency-aware Durability**

- **delay durability of writes**
- **make data durable before serving reads**

**good performance**

prevents out-of-order data across failures
strong consistency
Cross-client Monotonic Reads upon CAD
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client

\[
\begin{array}{ccc}
a_0 & a_1 & a_2 \\
\end{array}
\]
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client

Even in the presence of failures and across client sessions
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client

Even in the presence of failures and across client sessions

No existing model provides this guarantee except linearizability but not with high performance
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client

Even in the presence of failures and across client sessions

No existing model provides this guarantee except linearizability but not with high performance

CAD enables this property with high performance
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client

Even in the presence of failures and across client sessions
No existing model provides this guarantee except linearizability but not with high performance
CAD enables this property with high performance

Does not prevent staleness like many weaker models
Cross-client Monotonic Reads upon CAD

A read from a client guaranteed to return at least the latest state returned to a previous read from any client

Even in the presence of failures and across client sessions
No existing model provides this guarantee except linearizability but not with high performance
CAD enables this property with high performance

Does not prevent staleness like many weaker models
However, avoids out-of-order data, useful in many app scenarios e.g., location-sharing, twitter timelines
Outline

Introduction
Motivation
CAD and cross-client monotonic reads
ORCA design
Results
Summary and conclusion
ORCA
ORCA

Implementation of consistency-aware durability and cross-client monotonic reads in leader-based majority systems
ORCA

Implementation of consistency-aware durability and cross-client monotonic reads in leader-based majority systems

Leader-based systems (e.g., MongoDB, ZooKeeper)
- leader – a dedicated node
- others are followers
- writes flow through leader, establishes a single order
ORCA

Implementation of consistency-aware durability and cross-client monotonic reads in leader-based majority systems

Leader-based systems (e.g., MongoDB, ZooKeeper)

leader – a dedicated node
others are followers
writes flow through leader, establishes a single order

Majority

data is safe when persisted on majority nodes (e.g., 3 out of 5 servers)
ORCA Write Path
ORCA Write Path

Same as an eventually durable system

leader – S1

S2

S3

in memory

on disk

durable = on disk on a majority
ORCA Write Path

Same as an eventually durable system

write b

leader – S1

S2

S3

in memory

on disk

durable = on disk on a majority
ORCA Write Path

Same as an eventually durable system

```
write b

leader – S1: a b
S2: a
S3: a

in memory  a₀ on disk  durable = on disk on a majority
```
ORCA Write Path

Same as an eventually durable system

- leader – S1
- S2
- S3

Immediately acknowledge writes → high performance

- in memory
- a on disk
- durable = on disk on a majority
ORCA Write Path

Same as an eventually durable system

leader – S1

immediately acknowledge writes → high performance

replication and persistence in background

in memory  a0  on disk  durable = on disk on a majority
ORCA Write Path

Same as an eventually durable system

write b

leader – S1

S2

S3

immediately acknowledge writes → high performance

replication and persistence in background

in memory  a₀  on disk  durable = on disk on a majority
ORCA Read Path

leader – S1

S2

S3

in memory

on disk

durable = on disk on a majority
ORCA Read Path

leader – S1

S2

S3

in memory

on disk

durable = on disk on a majority
ORCA Read Path

**Durable-index** – index of the latest durable item in the system

- **leader** – S1
- **S2**
- **S3**

**durable = on disk on a majority**

- **in memory**
- **on disk**
ORCA Read Path

Durable-index – index of the latest durable item in the system
Update-index of item $i$ – index of the last update that modified $i$
ORCA Read Path

**Durable-index** – index of the latest durable item in the system

**Update-index of item** $i$ – index of the last update that modified $i$

**Durability check** – $i$ durable if update-index of $i \leq$ durable-index of system
**ORCA Read Path**

**Durable-index** – index of the latest durable item in the system  
**Update-index of item i** – index of the last update that modified i  
**Durability check** – i is durable if update-index of i ≤ durable-index of system

```
  durable-index: 1  
  a’s update-index: 1  
  a is durable
```

**leader – S1**  
```
  a
  b
```

**S2**  
```
  a
```

**S3**  
```
  a
```

- **in memory**
- **on disk**
- **durable = on disk on a majority**
ORCA Read Path

Durable-index – index of the latest durable item in the system
Update-index of item $i$ – index of the last update that modified $i$
Durability check – $i$ durable if update-index of $i \leq$ durable-index of system

durable-index: 1
a’s update-index: 1
a is durable
serve read immediately

leader – S1

S2

S3

read a

in memory
on disk
durable = on disk on a majority
ORCA Read Path

**Durable-index** – index of the latest durable item in the system

**Update-index of item i** – index of the last update that modified i

**Durability check** – i is durable if update-index of i ≤ durable-index of system

- durable-index: 1
- a’s update-index: 1
- a is durable
- serve read immediately

**leader – S1**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

**S2**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**S3**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**in memory**

**on disk**

**durable = on disk on a majority**
**ORCA Read Path**

**Durable-index** – index of the latest durable item in the system

**Update-index of item** $i$ – index of the last update that modified $i$

**Durability check** – $i$ is durable if update-index of $i \leq$ durable-index of system

<table>
<thead>
<tr>
<th>durable-index: 1</th>
<th>durability-index: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a’s update-index: 1</td>
<td>b’s update-index: 2</td>
</tr>
<tr>
<td>a is durable</td>
<td>b is not durable</td>
</tr>
</tbody>
</table>

Serve read immediately

**leader – S1**

- S1
  - a
  - b

- S2
  - a

- S3
  - a

<table>
<thead>
<tr>
<th>in memory</th>
<th>on disk</th>
</tr>
</thead>
</table>

**durable = on disk on a majority**
ORCA Read Path

**Durable-index** – index of the latest durable item in the system

**Update-index of item** $i$ – index of the last update that modified $i$

**Durability check** – $i$ durable if update-index of $i \leq$ durable-index of system

- durable-index: 1
  - a’s update-index: 1
  - a is durable
  - serve read immediately

- durable-index: 1
  - b’s update-index: 2
  - b is not durable
  - make b durable before serving

**leader – S1**

- a
- b

**S2**

- a

**S3**

- a

**read a**

- in memory

**durable = on disk on a majority**

**on disk**
**ORCA Read Path**

**Durable-index** – index of the latest durable item in the system

**Update-index of item i** – index of the last update that modified i

**Durability check** – i is durable if update-index of i ≤ durable-index of system

- durable-index: 1
  - a’s update-index: 1
  - a is durable
  - serve read immediately

- durable-index: 1
  - b’s update-index: 2
  - b is not durable
  - make b durable before serving

**leader** – S1

S2

S3

![Read a](read_a.png)

![Read b](read_b.png)

**in memory**

**on disk**

durable = on disk on a majority
ORCA Read Path

**Durable-index** – index of the latest durable item in the system

**Update-index of item** $i$ – index of the last update that modified $i$

**Durability check** – $i$ is durable if update-index of $i \leq$ durable-index of system

- **durable-index:** 1
  - a’s update-index: 1
  - a is durable
  - serve read immediately

- **durable-index:** 1
  - b’s update-index: 2
  - b is not durable
  - make b durable before serving

**leader – S1**

- **S2**
  - a

- **S3**
  - a

**read a**

**in memory**

**on disk**

**durable = on disk on a majority**
**ORCA Read Path**

**Durable-index** – index of the latest durable item in the system

**Update-index of item** $i$ – index of the last update that modified $i$

**Durability check** – $i$ is durable if update-index of $i \leq$ durable-index of system

- `durable-index: 1`
  - a’s update-index: 1
  - a is durable
  - serve read immediately

- `durable-index: 1`
  - b’s update-index: 2
  - b is not durable
  - make b durable before serving

**leader – S1**

- `S2` a
- `S3` a

**read a**

**durable = on disk on a majority**

<table>
<thead>
<tr>
<th>in memory</th>
<th>on disk</th>
</tr>
</thead>
</table>

19
ORCA Read Path

**Durable-index** – index of the latest durable item in the system

**Update-index of item** \( i \) – index of the last update that modified \( i \)

**Durability check** – \( i \) is durable if update-index of \( i \) ≤ durable-index of system

<table>
<thead>
<tr>
<th>leader – S1</th>
<th>durable-index: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a is durable</td>
<td></td>
</tr>
<tr>
<td>serve read immediately</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S2</th>
<th>a’s update-index: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S3</th>
<th>a’s update-index: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>durable-index: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>b’s update-index: 2</td>
</tr>
<tr>
<td>b is not durable</td>
</tr>
<tr>
<td>make b durable before serving</td>
</tr>
</tbody>
</table>

**durable = on disk on a majority**

- in memory
- on disk

---

19
ORCA Read Path

**Durable-index** – index of the latest durable item in the system
**Update-index of item i** – index of the last update that modified i
**Durability check** – i is durable if update-index of i ≤ durable-index of system

```
- durable-index: 1
- a’s update-index: 1
- a is durable
  - serve read immediately

- durable-index: 1
- b’s update-index: 2
- b is not durable
  - make b durable before serving
```

**leader – S1**

```
S1: a b
S2: a
S3: a
```

**read a**

```
S1: a b
S2: a
S3: a
```

**read b**

```
S1: a b
S2: a
S3: a
```

```
S1: a b
S2: a
S3: a
```

**in memory**

```
in memory
```

**on disk**

```
on disk
```

**durable = on disk on a majority**
ORCA Read Path

Durable-index – index of the latest durable item in the system
Update-index of item $i$ – index of the last update that modified $i$
Durability check – $i$ is durable if update-index of $i \leq$ durable-index of system

durable-index: 1
a’s update-index: 1
a is durable
serve read immediately

durable-index: 1
b’s update-index: 2
b is not durable
make b durable before serving

leader – S1

S2 a
S3 a

read a

in memory

leader – S2

on disk

on a majority

durable = on disk on a majority
**ORCA Read Path**

**Durable-index** – index of the latest durable item in the system

**Update-index of item** $i$ – index of the last update that modified $i$

**Durability check** – $i$ durable if update-index of $i \leq$ durable-index of system

- **durable-index:** 1
  - a’s update-index: 1
  - a is durable
  - serve read immediately

- **durable-index:** 1
  - b’s update-index: 2
  - b is not durable
  - make b durable before serving

**leader – S1**

- S2
- S3

read a

\[ \text{a} \]

\[ \text{a} \]

\[ \text{a} \]

**leader – S2**

\[ \text{a} \]

\[ \text{a} \]

\[ \text{b} \]

\[ \text{a} \]

\[ \text{b} \]

\[ \text{b} \]

\[ \text{b} \]

\[ \text{b} \]

\[ \text{b} \]
Cross-Client Monotonic Reads in ORCA
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers
- lagging followers could cause out-of-order states, CAD is not sufficient
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers

lagging followers could cause out-of-order states, CAD is not sufficient

leader – S1

S2

S3

S4

S5

a1 a2
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads
not scalable

Allow reads at followers
lagging followers could cause out-of-order states, CAD is not sufficient
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers
  lagging followers could cause out-of-order states, CAD is not sufficient

![Diagram showing cross-client monotonic reads](image)
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers
  lagging followers could cause out-of-order states, CAD is not sufficient

durability check fails
make durable
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers
lagging followers could cause out-of-order states, CAD is not sufficient

durability check fails
make durable
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads
not scalable

Allow reads at followers
lagging followers could cause out-of-order states, CAD is not sufficient

durability check fails
make durable
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads
not scalable

Allow reads at followers
lagging followers could cause out-of-order states, CAD is not sufficient

read a

leader – S1

S2

S3

S4

S5

durability check fails
make durable

20
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers

lagging followers could cause out-of-order states, CAD is not sufficient

durability check fails make durable
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers

tagging followers could cause out-of-order states, CAD is not sufficient

durability check fails make durable

durability check passes but doesn’t know a later version has been served!
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers
  lagging followers could cause out-of-order states, CAD is not sufficient

durability check fails
make durable

durability check passes but doesn’t know a later version has been served!
Cross-Client Monotonic Reads in ORCA

If reads restricted to leader, CAD provides cross-client monotonic reads not scalable

Allow reads at followers
lagging followers could cause out-of-order states, CAD is not sufficient

durability check fails make durable

Additional mechanisms: Active sets (lease-based mechanism), not in this talk…
Outline

Introduction
Motivation
CAD and cross-client monotonic reads
ORCA design
Results
Summary and conclusion
Evaluation
Evaluation

Implemented in ZooKeeper
Evaluation

Implemented in ZooKeeper

Evaluate different durability models in isolation
  compare CAD against immediate and eventual durability
Evaluation

Implemented in ZooKeeper

Evaluate different durability models in isolation
  compare CAD against immediate and eventual durability

Evaluate overall system performance
  ORCA against strong and weakly consistent ZooKeeper
CAD Durability Layer Performance
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

Write Latency Distribution

CDF

Latency (us)

immediate  eventual  cad
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

Write Latency Distribution

CDF

Latency (us)

0 500 1000 1500

immediate eventual cad
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

CAD writes faster than immediate durability
CAD matches performance of eventual
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

CAD writes faster than immediate durability
CAD matches performance of eventual

Write Latency Distribution

Read Latency Distribution
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

CAD writes faster than immediate durability
CAD matches performance of eventual

Write Latency Distribution

Read Latency Distribution
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

CAD writes faster than immediate durability
CAD matches performance of eventual

Write Latency Distribution

Read Latency Distribution

CDF

Latency (us)

reads queued behind writes

immediate eventual cad

immediate eventual cad

0 500 1000 1500

0 500 1000 1500

0 20 40 60 80 100

0 20 40 60 80 100

0 20 40 60 80 100
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

Write Latency Distribution

- CAD writes faster than immediate durability
- CAD matches performance of eventual

Read Latency Distribution

- Most reads in CAD fast
- Only 5% slow due to synchronous ops

- Cad writes queued behind writes
CAD Durability Layer Performance

YCSB-A: 50% W, 50% R

- CAD writes faster than immediate durability
- CAD matches performance of eventual

**Write Latency Distribution**
- CDF
- Immediate
- Eventual
- CAD

**Read Latency Distribution**
- CDF
- Immediate
- Eventual
- CAD

- Most reads in CAD fast
- Only 5% slow due to synchronous ops

CAD performs similar to eventual and is faster than immediate
ORCA System Performance

Strong-ZK – uses **immediate** durability, reads only at **leader**
Weak-ZK – uses **eventual** durability, reads at many nodes
ORCA – uses **CAD**, reads at many nodes
ORCA System Performance

Strong-ZK – uses immediate durability, reads only at leader
Weak-ZK – uses eventual durability, reads at many nodes
ORCA – uses CAD, reads at many nodes

Throughput (KOps/s)
ORCA System Performance

Strong-ZK – uses **immediate** durability, reads only at **leader**
Weak-ZK – uses **eventual** durability, reads at many nodes
ORCA – uses **CAD**, reads at many nodes

Strong-ZK performs poorly due to immediate durability and leader-restricted reads
## ORCA System Performance

Strong-ZK – uses **immediate** durability, reads only at leader

Weak-ZK – uses **eventual** durability, reads at many nodes

ORCA – uses CAD, reads at many nodes

### Throughput (KOps/s)

<table>
<thead>
<tr>
<th></th>
<th>strong-ZK</th>
<th>weak-ZK</th>
<th>orca</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>3.78</td>
<td>3.28</td>
<td><strong>F</strong></td>
</tr>
<tr>
<td>50% R</td>
<td>50% W</td>
<td></td>
<td>66.7% R</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>2.09</td>
<td>1.97</td>
<td></td>
</tr>
<tr>
<td>95% R</td>
<td>5% W</td>
<td></td>
<td>33.3% W</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>2.09</td>
<td>1.75</td>
<td></td>
</tr>
<tr>
<td>95% R</td>
<td>5% W</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>F</strong></td>
<td>3.44</td>
<td>3.04</td>
<td></td>
</tr>
</tbody>
</table>

Strong-ZK performs poorly due to immediate durability and leader-restricted reads.

Weak-ZK performs well due to eventual durability and scalable reads.
ORCA System Performance

Strong-ZK – uses immediate durability, reads only at leader
Weak-ZK – uses eventual durability, reads at many nodes
ORCA – uses CAD, reads at many nodes

Strong-ZK performs poorly due to immediate durability and leader-restricted reads
Weak-ZK performs well due to eventual durability and scalable reads
ORCA adds little overheads compared to weak-ZK
reads that access non-durable data
ORCA System Performance

Strong-ZK – uses immediate durability, reads only at leader
Weak-ZK – uses eventual durability, reads at many nodes
ORCA – uses CAD, reads at many nodes

Strong-ZK performs poorly due to immediate durability and leader-restricted reads
Weak-ZK performs well due to eventual durability and scalable reads
ORCA adds little overheads compared to weak-ZK
reads that access non-durable data
More experiments in the paper…

Evaluation
- correctness testing using a cluster crash-testing framework
- geo-replicated setting
- micro-benchmarks

Application case studies
- location-tracking
- social-media timeline
Summary and Conclusions
Summary and Conclusions

Surprisingly, durability models are overlooked
Summary and Conclusions

Surprisingly, durability models are overlooked
Immediate durability enables strong consistency but is slow
Summary and Conclusions

Surprisingly, durability models are overlooked
Immediate durability enables strong consistency but is slow
Eventual durability is fast but only enables weak consistency
Summary and Conclusions

Surprisingly, durability models are overlooked
Immediate durability enables strong consistency but is slow
Eventual durability is fast but only enables weak consistency
CAD – consistency-aware durability, a new way of thinking about durability enables both strong consistency and high performance
Summary and Conclusions

Surprisingly, durability models are overlooked

Immediate durability enables strong consistency but is slow

Eventual durability is fast but only enables weak consistency

CAD – consistency-aware durability, a new way of thinking about durability

enables both strong consistency and high performance

CAD is useful for many deployments that currently adopt eventual durability
Summary and Conclusions

Surprisingly, durability models are overlooked

Immediate durability enables strong consistency but is slow

Eventual durability is fast but only enables weak consistency

CAD – consistency-aware durability, a new way of thinking about durability
  enables both strong consistency and high performance
  CAD is useful for many deployments that currently adopt eventual durability

Consistency and performance are seemingly at odds – by carefully examining the underlying layer, achieve both
Summary and Conclusions

Surprisingly, durability models are overlooked
Immediate durability enables strong consistency but is slow
Eventual durability is fast but only enables weak consistency

CAD – consistency-aware durability, a new way of thinking about durability
  enables both strong consistency and high performance
  CAD is useful for many deployments that currently adopt eventual durability

Consistency and performance are seemingly at odds – by carefully examining the underlying layer, achieve both

Thank you!