Incremental Consistency Guarantees
For Replicated Objects

Rachid Guerraoui, Matej Pavlovic, Dragos-Adrian Seredinschi
Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
SOCIAL MEDIA APPLICATION

Number of recent events on the user’s timeline

Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
Strong Consistency

- Returns the **correct** data \(22\)
- Latency: \(~200\) ms
- Can become **unavailable**

[CAP], [PACELC]
Strong Consistency
- Returns the **correct** data
- Latency: \(~100\) ms
- Can become **unavailable**
  
  [CAP], [PACELC]

Weak Consistency
- Latency: \(~100\) ms
- High availability
- Allows inconsistencies: can return
  
  \(22\) or \(20\) or \(17\)
Incremental Consistency Guarantees

Dragos-Adrian Seredinschi

- Latency: ~100 ms
- High availability
- Allows inconsistencies: can return incorrect data
- Latency: ~200 ms
- Can become unavailable

Neither model is ideal!

Number of recent events on the user’s timeline

Social Media Application

Strong Consistency

Weak Consistency

Neither model is ideal!
Incremental Consistency Guarantees
Dragos-Adrian Seredinschi

- Latency: \(~100\, \text{ms}\)
- High availability
- Allows inconsistencies: can return correct data
- Latency: \(~200\, \text{ms}\)
- Can become unavailable

Strong Consistency  Weak Consistency

Neither model is ideal!

We use both models.
Multiple models

1. **Weak consistency**
   - 100ms → 20

2. **Strong consistency**
   - 300ms → 22
Multiple models

1. **Weak consistency**
   100ms → 20

2. **Strong consistency**
   300ms → 22

Increasingly many systems expose multiple consistency models:

- Dynamo [SOSP’07]
- Pileus [SOSP’13]
Multiple models

1. **Weak consistency**
   - 100ms → 20

2. **Strong consistency**
   - 300ms → 22

Issues

1. **Send multiple requests?**
2. **How to leverage individual responses?**
3. **Semantics?**
4. ...
Multiple models

1. **Weak consistency**
   - 100ms → 20

2. **Strong consistency**
   - 300ms → 22

Issues

1. **Send multiple requests?**

2. **How to leverage individual responses?**

3. **Semantics?**

4. **...**

Increasingly many systems expose multiple consistency models:

- Dynamo [SOSP’07]
- Cassandra
- SimpleDB
- RiakKV
- Pileus [SOSP’13]

Problem

How do you program with
- inconsistencies?
- multiple values?
ABSTRACTION
FOR REPLICATED OBJECTS

SOCIAL MEDIA APPLICATION

Weak Consistency

Strong Consistency

???? Consistency

Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
Incremental Consistency Guarantees
Dragos-Adrian Seredinschi

Abstraction for Replicated Objects

Weak Consistency

Strong Consistency

??? Consistency

Social Media Application

Correctable
Incremental Consistency Guarantees (ICG) provides correctable consistency for replicated objects. This framework offers a trade-off between weak consistency and strong consistency, allowing for incremental consistency guarantees (ICG) that balance performance and correctness in social media applications.
Correctables / Design

- Starting point: Promises
  - Placeholders for values
  - Becoming mainstream
Correctables / Design

- Starting point: Promises
- Placeholders for values
- Becoming mainstream

Starting point: Promises

Placeholders for values

Becoming mainstream
Correctables / Design

» Starting point: Promises
  » Placeholders for values
  » Becoming mainstream

Futures and Promises

Scala

**Incremental Consistency Guarantees**

Dragos-Adrian Seredinschi

**Correctables / Design**

» Starting point: Promises
  » Placeholders for values
  » Becoming mainstream

Futures for C++11 at Facebook

Google Guava
Core libraries for Java & Android

PROGRESSIVELY STRONGER CONSISTENCY (INCREMENTAL)

PROGRESSIVELY HIGHER LATENCY

Correctables / Design

» Starting point: Promises
  » Placeholders for values
  » Becoming mainstream

Futures for C++11 at Facebook

Google Guava
Core libraries for Java & Android

**PROGRESSIVELY STRONGER CONSISTENCY (INCREMENTAL)**

**PROGRESSIVELY HIGHER LATENCY**
Consistency Models are **Complementary**

- **Weak Consistency**:
  - Fast
  - (Often correct)

- **Strong Consistency**:
  - Slower
  - (Correct with certainty)

---

Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
Consistency Models are Complementary

Consistency

Weak Consistency: ★ Fast ★ (Often correct)

Strong Consistency: ★ Slower ★ (Correct with certainty)

So what?
Consistency Models are **Complementary**

<table>
<thead>
<tr>
<th>Consistency Models</th>
<th>Weak Consistency</th>
<th>Strong Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>★ Fast</td>
<td>★ Slower</td>
<td></td>
</tr>
<tr>
<td>★ (Often correct)</td>
<td>★ (Correct with certainty)</td>
<td></td>
</tr>
</tbody>
</table>

**So what?**

**Latency optimizations**
Speculating with Correctables

SOCIAL MEDIA APPLICATION

read timeline

CORRECTABLE
Speculating with Correctables

SOCIAL MEDIA APPLICATION

value\(^1\)  
read timeline  
Weak Consistency

Latency gap: \(\sim 100\) ms

value\(^2\)  
Strong Consistency

CORRECTABLE

Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
Speculating with Correctables

SOCIAL MEDIA APPLICATION

value¹

Speculative execution

Latency gap: ~100 ms

value²

read timeline

• value¹ is often correct
  [Existential Consistency. SOSP’15]
  [PBS. VLDB 5(8)’12]

• Speculatively execute any further steps
e.g., prefetch dependent data

CORRECTABLE

Verify based on value²

Lower latency of strong consistency

Latency gap: ~100 ms
Speculating with Correctables

**SOCIAL MEDIA APPLICATION**

- **value\(^1\)**
  - Weak Consistency
  - read timeline
  - value\(^1\) is often correct
    - [Existential Consistency. SOSP’15]
    - [PBS. VLDB 5(8)’12]
  - Speculatively execute any further steps
e.g., prefetch dependent data

- **value\(^2\)**
  - Strong Consistency
  - Verify based on value\(^2\)

**CORRECTABLE**

Lower latency of strong consistency

Latency gap: \(~100\) ms
Traditional operation:

Strongly-consistent timeline

200 ms

Fetch timeline items

100 ms

300 ms
**Traditional** operation:

Strongly-consistent timeline

200 ms

Fetch timeline items

100 ms

300 ms

**Speculative** operation with ICG:

value $^1$

100 ms

value $^2$

100 ms

Fetch timeline items

100 ms
**Traditional operation:**

Strongly-consistent timeline

- 200 ms
- Fetch timeline items
- 100 ms
- 300 ms

**Speculative operation with ICG:**

- **Value**¹: 100 ms
- **Value**²: 100 ms
- Fetch timeline items: 100 ms
- **Value**¹ matches **Value**²: yes
- 200 ms
**Traditional** operation:

Strongly-consistent timeline

- 200 ms

Fetch timeline items

- 100 ms

300 ms

**Speculative** operation with ICG:

- 100 ms

value\(^1\)

value\(^2\)

Fetch timeline items

- 100 ms

value\(^1\) matches value\(^2\)

yes

200 ms

- 100 ms

Re-fetch

no

300 ms

- 100 ms
Speculation case-study

➤ Application: Twissandra
➤ Workload generated via YCSB
➤ Clients in Ireland
➤ Geo-replication on Amazon’s EC2
Speculation case-study

- Application: Twissandra
- Workload generated via YCSB
- Clients in Ireland
- Geo-replication on Amazon’s EC2

Incremental Consistency Guarantees for Replicated Objects

Rachid Guerraoui, Matej Pavlovic, and Dragos-Adrian Seredinschi

School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{rachid.guerraoui, matej.pavlovic, dragos-adrian.seredinschi}@epfl.ch

check the paper

- Advertising System
  — Speculation case-study
- Ticket-selling System
  — Exploiting application semantics
- Overheads evaluation & Optimizations
- Latency gaps between consistency models
Decreasing latency of strong consistency

What is the latency of the `fetch_timeline()` operation?
Decreasing latency of strong consistency

What is the latency of the `fetch_timeline()` operation?

**Baseline**
Read using a quorum of 2/3 replicas

**ICG**
1. *Weak:* Read with 1/3 replicas
2. *“Strong”:* Read with quorum of 2/3 replicas
Decreasing latency of strong consistency

What is the latency of the `fetch_timeline()` operation?

Workload A (50:50 read/write)  Workload B (95:5 read/write)  Workload C (read-only)

**Baseline**
Read using a quorum of 2/3 replicas

**ICG**
1. *Weak:* Read with 1/3 replicas
2. *Strong:* Read with quorum of 2/3 replicas

<table>
<thead>
<tr>
<th>Workload A (50:50 read/write)</th>
<th>Workload B (95:5 read/write)</th>
<th>Workload C (read-only)</th>
</tr>
</thead>
</table>

---

Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
Decreasing latency of strong consistency

What is the latency of the `fetch_timeline()` operation?

**Baseline**
Read using a quorum of 2/3 replicas

**ICG**

1. Weak: Read with 1/3 replicas
2. “Strong”: Read with quorum of 2/3 replicas
Decreasing latency of strong consistency

What is the latency of the `fetch_timeline()` operation?

- **Latency decrease by 40%**
- **Throughput drop by 6%**
- **Same consistency model (2/3 replicas)**

Workload A (50:50 read/write)
- **Baseline**
  - Read using a quorum of 2/3 replicas

Workload B (95:5 read/write)
- **ICG**

Workload C (read-only)

**ICG**
1. **Weak**: Read with 1/3 replicas
2. **“Strong”**: Read with quorum of 2/3 replicas
Conclusion

The Correctables abstraction enables you to:

1. Leverage consistency models incrementally
2. Lower latency of strong consistency
backup slides
Speculation // Syntactic sugar

```javascript
1 invoke(read(...))
2 .speculate(speculationFunc[, abortFunc])
3 .setCallbacks(onFinal = (res) => deliver(res))
```

**Listing 3:** Generic speculation with Correctables. The square brackets indicate that `abortFunc` is optional.
Legacy code vs. Correctables

Listing 1: Different consistency guarantees in Reddit [13], as an example of tight coupling between applications and storage. Developers must manually handle the cache and the backend.

```
from pylons import app_globals as g  # cache access
from r2.lib.db import queries        # backend access

def user_messages(user, update = False):
    key = messages_key(user._id)
    trees = g.permacache.get(key)
    if not trees or update:
        trees = user_messages_nocache(user)
        g.permacache.set(key, trees)  # cache coherence
    return trees

def user_messages_nocache(user):
    # Just like user_messages, but avoiding the cache...
```

Listing 2: Reddit code rewritten using Correctables.

```
def user_messages(user, strong = False):
    key = messages_key(user._id)
    # coherence handled by invoke* functions in bindings
    if strong: return invokeStrong(get(key))
    else: return invokeWeak(get(key))
```

Correctable

Semantics

Execution

Details
**Legacy code vs. Correctables**

```python
from pylons import app_globals as g  # cache access
from r2.lib.db import queries        # backend access

def user_messages(user, update = False):
    key = messages_key(user._id)
    trees = g.permacache.get(key)
    if not trees or update:
        trees = user_messages_nocache(user)
        g.permacache.set(key, trees)  # cache coherence
    return trees

def user_messages_nocache(user):
    # Just like user_messages, but avoiding the cache...
```

**Listing 1:** Different consistency guarantees in Reddit [13], as an example of tight coupling between applications and storage. Developers must manually handle the cache and the backend.

```python
invoke(getLatestNews()).setCallbacks(
    onUpdate = (items) => refreshDisplay(items))

```

**Listing 6:** Progressive display of news items using Correctables. The `refreshDisplay` function triggers with every update on the news items.

```
def user_messages(user, strong = False):
    key = messages_key(user._id)
    # coherence handled by invoke* functions in bindings
    if strong: return invokeStrong(get(key))
    else: return invokeWeak(get(key))
```

**Listing 2:** Reddit code rewritten using Correctables.
Overheads

- Cassandra
- YCSB workload, various configurations
- Client in Ireland
- Replicas in Virginia, Frankfurt, and Ireland
Overheads

- Cassandra
- YCSB workload, various configurations
- Client in Ireland
- Replicas in Virginia, Frankfurt, and Ireland

- ZooKeeper queue implementation
- Wasteful implementation (by default)
- We were able to improve — negative overhead
Exploiting application semantics

➤ Ticket selling application
  ➤ Implemented through a ZooKeeper queue
  ➤ Buy ticket = dequeue operation
Exploiting application semantics

- Ticket selling application
  - Implemented through a ZooKeeper queue
  - Buy ticket = dequeue operation
Divergence between weak and strong consistency

![Graph showing divergence between weak and strong consistency for different workloads. The graph plots %Divergence against #Total client threads. Workload A-Latest, Workload A-Zipfian, Workload B-Latest, and Workload B-Zipfian are compared. The graph indicates extremley high divergence, unusual, and typical range of 0.1% to 5%.](image)

- Extremely high
- Unusual
- 0.1% — 5% typical range
Latency gaps between consistency models

Workload A (50:50 read/write)

Workload B (95:5 read/write)

Workload C (read-only)

Average Read Latency (ms)

Latency gap

99th %ile latency

CT (R=1)

C^2 (R=2)

CC^2 preliminary (R=1)

CC^2 final (R=2)

Throughput (ops/sec)

Average Latency (ms)

Latency gap

99th %ile latency

CC preliminary

CC final

C

Leader in IRL

Leader in VRG

Follower (FRK)

Leader (IRL)

Follower (IRL)

Leader (VRG)

Client connection:
Efficiency of Multiple Responses

SOCIAL MEDIA APPLICATION

read timeline

value

CORRECTABLE

value


Efficiency of Multiple Responses

SOCIAL MEDIA APPLICATION

read timeline

value¹

value²

CORRECTABLE

Binding

Replicated Storage

cassandra
Efficiency of Multiple Responses

SOCIAL MEDIA APPLICATION

Binding

Replicated Storage

Request

CORRECTABLE

Weak consistency

Strong consistency

Response (preliminary)

Response (final)

Coordination (quorum gathering)

value\(^1\)

value\(^2\)

read timeline

value\(^1\)

value\(^2\)

Efficiency of Multiple Responses

SOCIAL MEDIA APPLICATION

Binding

Replicated Storage

Request

CORRECTABLE

Weak consistency

Strong consistency

Response (preliminary)

Response (final)

Coordination (quorum gathering)

value\(^1\)

value\(^2\)

read timeline

value\(^1\)

value\(^2\)
Correctables / Library

Desktop Application
Web Frontend
Mobile App
Caching Daemon

invoke

Correctables
LIBRARY

API

Correctable

RPC (Weak / Strong)

Consistency-based
System-specific
Interface

Desktop Application
Web Frontend
Mobile App
Caching Daemon

Correctables
LIBRARY

RPC

API

Correctable

RPC

Invoke

Correctables
LIBRARY
Correctables / **Library**
Incremental Consistency Guarantees

Dragos-Adrian Seredinschi

E-mail
Calendar
Social networks
Online shopping
Ad serving
News reading
Collaborative editing

Performance is a second-order concern

Computation on static content
Cold data analysis
Disconnected operations in mobile applications

Semantics allow to bypass the need for strong consistency
Replicated storage systems

Richness of consistency models (CMs)

- 1 CM (strong)
- 2 CMs (weak and strong)
- Caching (client & server)
- Dynamic quorums
- 6 CMs
- 10 CMs

CAP theorem

Time

SQLite

Dynamo [SOSP’07]
Pileus [SOSP’13]

MySQL

PostgreSQL

SimpleDB

App Engine

Redis

Cassandra

RiakKV

MEMCACHED

Incremental Consistency Guarantees
Dragos-Adrian Seredinschi
Replicated storage systems

Richness of consistency models (CMs)

- 1 CM (strong)
- 2 CMs (weak and strong)
- Caching (client & server)
- dynamic quorums
- 6 CMs
- 10 CMs

1 CM
- MySQL (strong)
- PostgreSQL

2 CMs
- Dynamo [SOSP'07]
- SimpleDB
- Memcached

Caching (client & server)
- SQLite
- Redis

dynamic quorums
- Cassandra
- Riak

CAP theorem

No single consistency model is ideal → expose multiple choices