Making Geo-Replicated Systems Fast as Possible Consistent when Necessary

Cheng Li†, Daniel Porto†§, Allen Clement†
Johannes Gehrke‡, Nuno Preguiça§, Rodrigo Rodrigues§

Max Planck Institute for Software Systems†,
CITI / Universidade Nova de Lisboa§, Cornell University‡
Higher latency => Less money

[source: E. Schurman and J. Brutlag, “Performance Related Changes and their User Impact”. Talk at Velocity ‘09]
Geo-replication is needed!

- Geo-replication is used by major providers of Internet services.
  - e.g., Google, Amazon, Facebook, etc
Consistency or performance?

**Strong consistency**
- e.g., Paxos [TOCS′98]
- Pros: *Natural semantics*
- Cons: *High latency*

**Eventual consistency**
- e.g., Dynamo [SOSP′07], Bayou [SOSP′95]
- Pros: *Low latency*
- Cons: *Undesirable behaviors*

Can we build geo-replicated systems that are both fast and consistent?
Outline

• Mixing strong and eventual consistency in a single system

• Transforming applications to safely leverage eventual consistency when possible

• Evaluation
Balance strong/eventual consistency

Strong consistency

- R1
- R2
- R3

Eventual consistency

- A1
- A2
- B1
- B2
- B3
Balance strong/eventual consistency

**Strong consistency**

R1 → R2 → R3

**Eventual consistency**

A1 → A2 → B1, B2, B3
Balance strong/eventual consistency

**Strong consistency** RedBlue

- Low latency of eventual consistency when possible
- Coordination for strong consistency only when necessary

**Eventual consistency**
Gemini coordination system

Cross-site communication
Gemini coordination system

Cross-site communication

Alice

Bob

Storage engine

Coordinator

R3

R1

A1

B1

B2

R2

B4

A2

B1

B2

R2

R3

R3

B4

A1

R1

Storage engine

Coordinator

Cross-site communication
A RedBlue consistent bank system
**A RedBlue consistent bank system**

- **Problem:** Different execution orders lead to divergent state.
- **Cause:** `accrueinterest` doesn’t commute with `deposit`.
- **Implication:** Convergence requires **Red**, but **Red** is slow.

```c
float balance, interest;

void deposit(float m)
{
    balance = balance + m;
}

double accrueinterest()
{
    float delta = balance * interest;
    balance = balance + delta;
}

void withdraw(float m)
{
    if (balance - m >= 0)
        balance = balance - m;
    else
        printf("Error\n");
}
```
Outline

• Mixing strong and eventual consistency in a single system

• Transforming applications to safely leverage eventual consistency when possible

• Evaluation
Problem of replicating operations

Initial: balance = 100, interest = 0.05

Alice in EU

100

\textit{deposit}(20): +20

120

\textit{accrueinterest}(): +6

126

Bob in US

100

\textit{accrueinterest}(): +5

125

Two different decisions made by the same operation
Generator/Shadow operation

- Intuitively, the execution of *accruing_interest* can be divided into:
  - A generator operation
    - decides *how much interest to be accrued*
    - has *no side effects*
  - A shadow operation
    - adds *the decided interest to the balance*
Generate once, shadow everywhere

1. Generator makes a decision $d$ based on a local state $S$.
2. Generator produces a colored shadow $h(d,S)$.
3. Shadow is applied at all sites.

Cross-site communication
## Bank generator/shadow operations

### Original/Generator operation

<table>
<thead>
<tr>
<th>deposit(float m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>balance = balance + m;</td>
</tr>
</tbody>
</table>
| }

<table>
<thead>
<tr>
<th>accrueinterest()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>float delta=balance \times \text{interest};</td>
</tr>
<tr>
<td>balance=balance + delta;</td>
</tr>
</tbody>
</table>
| }

<table>
<thead>
<tr>
<th>withdraw(float m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>if(balance-m&gt;=0)</td>
</tr>
<tr>
<td>balance=balance - m;</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>print “Error”</td>
</tr>
</tbody>
</table>
| }

### Shadow operation

<table>
<thead>
<tr>
<th>deposit’(float m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>balance = balance + m;</td>
</tr>
</tbody>
</table>
| }

<table>
<thead>
<tr>
<th>accrueinterest’(float delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>balance=balance + delta;</td>
</tr>
</tbody>
</table>
| }

<table>
<thead>
<tr>
<th>withdrawAck’(float m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>{ balance=balance - m;</td>
</tr>
</tbody>
</table>
| }

| withdrawFail’()|
Bank generator/shadow operations

Original/Generator operation

```c
void deposit(float m){
    balance = balance + m;
}
```

```c
void accrueinterest(float delta){
    balance = balance + delta;
}
```

```c
void withdraw(float m){
    if(balance-m>=0)
        balance = balance - m;
    else
        print “Error”
}
```

Shadow operation

```c
void deposit’(float m){
    balance = balance + m;
}
```

```c
void accrueinterest’(float delta){
    balance = balance + delta;
}
```

```c
void withdrawAck’(float m){
    balance = balance - m;
}
```

```c
void withdrawFail’(){
}
```

All four shadow banking operations commute with each other!
Fast and consistent bank

Initial: balance = 100, interest = 0.05

Alice in EU

Bob in US

deposit(20) : +20

accrueinterest() : +5

Generator op

Shadow op

100

+20

120

+20

125

100

125

105
Not so fast ...

Initial: \(balance = 100, \ interest = 0.05\)

Alice in EU

100

\(\text{deposit}(20) : +20\)

+20

120

+5

125

\(\text{withdraw}(100): -100\)

Bob in US

100

\(\text{accrueinterest}(): +5\)

+5

105

+20

125

\(\text{withdraw}(80): -80\)
Not so fast ...

- **Problem**: Different execution orders lead to a negative balance.

- **Cause**: Blue operations that potentially break invariants execute without coordination.

- **Implication**: We must label successful withdrawal (withdrawAck) as Red.
Which must be Red or can be Blue?

- Ensuring state convergence
- Ensuring invariant preservation

a shadow operation \( u \)

commutes with all others?

Yes → breaks invariants?

Yes → Red

No → Yes

No → Blue

Ensuring invariant preservation
Key ideas so far

• RedBlue consistency combines strong and eventual consistency into a single system.

• The decomposition of generator/shadow operations expands the space of possible Blue operations.

• A simple rule for labeling is provably state convergent and invariant preserving.
Evaluation
Questions

• How common are Blue operations?

• Does RedBlue consistency improve user-observed latency?

• Does throughput scale with the number of sites?
Questions

• How common are Blue operations?
• Does RedBlue consistency improve user-observed latency?
• Does throughput scale with the number of sites?
Case studies

• Applications:
  – Two e-commerce benchmarks: TPC-W, RUBiS
  – One social networking app: Quoddy

<table>
<thead>
<tr>
<th>Apps</th>
<th># Original update txns</th>
<th># Blue/Red update ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-W</td>
<td>7</td>
<td>0/7</td>
</tr>
<tr>
<td>RUBiS</td>
<td>5</td>
<td>0/5</td>
</tr>
<tr>
<td>Quoddy</td>
<td>4</td>
<td>0/4</td>
</tr>
</tbody>
</table>
Case studies

- Applications:
  - Two e-commerce benchmarks: TPC-W, RUBiS
  - One social networking app: Quoddy

<table>
<thead>
<tr>
<th>Apps</th>
<th># Original update txns</th>
<th># Blue/Red update ops</th>
<th># Shadow ops</th>
<th># Blue/Red update ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-W</td>
<td>7</td>
<td>0/7</td>
<td>16</td>
<td>14/2</td>
</tr>
<tr>
<td>RUBiS</td>
<td>5</td>
<td>0/5</td>
<td>9</td>
<td>7/2</td>
</tr>
<tr>
<td>Quoddy</td>
<td>4</td>
<td>0/4</td>
<td>4</td>
<td>4/0</td>
</tr>
</tbody>
</table>
How common are **Blue operations**?

Runtime **Blue/Red** ratio in different applications with different workloads:

<table>
<thead>
<tr>
<th>Apps</th>
<th>workload</th>
<th>Originally</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blue (%)</td>
</tr>
<tr>
<td>TPC-W</td>
<td>Browsing mix</td>
<td>96.0</td>
</tr>
<tr>
<td></td>
<td>Shopping mix</td>
<td>85.0</td>
</tr>
<tr>
<td></td>
<td>Ordering mix</td>
<td>63.0</td>
</tr>
<tr>
<td>RUBiS</td>
<td>Bidding mix</td>
<td>85.0</td>
</tr>
<tr>
<td>Quoddy</td>
<td>a mix with 15% update</td>
<td>85.0</td>
</tr>
</tbody>
</table>
How common are Blue operations?

Runtime Blue/Red ratio in different applications with different workloads:

<table>
<thead>
<tr>
<th>Apps</th>
<th>workload</th>
<th>Originally</th>
<th>With shadow ops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blue (%)</td>
<td>Red (%)</td>
</tr>
<tr>
<td>TPC-W</td>
<td>Browsing mix</td>
<td>96.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>Shopping mix</td>
<td>85.0</td>
<td>15.0</td>
</tr>
<tr>
<td></td>
<td>Ordering mix</td>
<td>63.0</td>
<td>37.0</td>
</tr>
<tr>
<td>RUBiS</td>
<td>Bidding mix</td>
<td>85.0</td>
<td>15.0</td>
</tr>
<tr>
<td>Quoddy</td>
<td>a mix with 15% update</td>
<td>85.0</td>
<td>15.0</td>
</tr>
</tbody>
</table>

The vast majority of operations are Blue.
Questions

• How common are Blue operations?

• Does RedBlue consistency improve user-observed latency?

• Does throughput scale with the number of sites?
Experimental setup

• Experiments with:
  – TPC-W, RUBiS and Quoddy

• Deployment in Amazon EC2
  – spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore)
  – locating users in all five sites and directing their requests to closest server
Experimental setup

• Experiments with:
  – TPC-W, RUBiS and Quoddy

• Deployment in Amazon EC2
  – spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore)
  – locating users in all five sites and directing their requests to closest server
Does RedBlue consistency improve user-observed latency?

![Diagram showing average latency for users at all five sites](image)

- US-East
- US-West
- Ireland
- Brazil
- Singapore

Average latency for users at all five sites

10/9/2012 Cheng Li@OSDI’12
Does throughput scale with the number of sites?

Peak throughput for different deployments
Conclusion

• RedBlue consistency allows strong consistency and eventual consistency to coexist.

• Generator/shadow operation extends the space of fast operations.

• A precise labeling methodology allows for systems to be fast and behave as expected.

• Experimental results show our solution improves both latency and throughput.
Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary

THANK YOU!