GEMINI: MAKING GEO-REPLICATED SYSTEMS FAST AS POSSIBLE, CONSISTENT WHEN NECESSARY

C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguica, and R. Rodrigues

Presented by Chun-Yu Chen
Outline

• Introduction
• RedBlue Consistency
• Gemini
• Evaluation
• Conclusions
Geo-Replication
Consistency vs. Performance

• Strong Consistency
  o Ex. Paxos
  o Always consistent, but long latency

• Eventual Consistency
  o Ex. Bayou
  o Low latency, but state could diverge
Design Goal

• Maintain meaningful consistency while not sacrificing performance
  o Allow multiple levels of consistency

• Low latency of eventual consistency when possible
• Coordination for strong consistency when necessary

• How to determine what to use or separate them?
RedBlue Operations

• Blue Operations
  o Execute locally
  o Lazily replicated in an eventually consistent manner

• Red Operations
  o Cross-site coordination
  o Serialized with each other
RedBlue Order

**Definition 1 (RedBlue order).** Given a set of operations $U = B \cup R$, where $B \cap R = \emptyset$, a RedBlue order is a partial order $O = (U, \prec)$ with the restriction that $\forall u, v \in R$ such that $u \neq v$, $u \prec v$ or $v \prec u$ (i.e., red operations are totally ordered).

- Either Red or Blue,
- Cannot be both Red and Blue
- Red operations are totally ordered
  (No concurrent red operations)
Causal Serialization

**Definition 2** (Causal serialization). Given a site $i$, $O_i = (U, \prec)$ is an $i$-causal serialization (or short, a causal serialization) of RedBlue order $O = (U, \prec)$ if (a) $O_i$ is a linear extension of $O$ (i.e., $\prec$ is a total order compatible with the partial order $\prec$), and (b) for any two operations $u, v \in U$, if $\text{site}(v) = i$ and $u \prec v$ in $O_i$, then $u \prec v$.

<table>
<thead>
<tr>
<th>Alice in EU</th>
<th>Bob in US</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$S_0'$</td>
</tr>
<tr>
<td>$\triangle a_1$</td>
<td>$\triangle b_1$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_1'$</td>
</tr>
<tr>
<td>$\triangle b_1$</td>
<td>$\triangle a_1$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_2'$</td>
</tr>
<tr>
<td>$\star b_2$</td>
<td>$\star b_2$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$S_3'$</td>
</tr>
<tr>
<td>$\triangle a_2$</td>
<td>$\triangle b_3$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$S_4'$</td>
</tr>
<tr>
<td>$\star a_3$</td>
<td>$\star a_2$</td>
</tr>
<tr>
<td>$S_5$</td>
<td>$S_5'$</td>
</tr>
<tr>
<td>$\triangle b_3$</td>
<td>$\star a_3$</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$S_6'$</td>
</tr>
<tr>
<td>$\star b_4$</td>
<td>$\star b_4$</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$S_7'$</td>
</tr>
</tbody>
</table>

- All a’s are in order
- All b’s are in order
- All red operations are in order
RedBlue Consistency

**Definition 3** (RedBlue consistency). A replicated system is $O$-RedBlue consistent (or short, RedBlue consistent) if each site $i$ applies operations according to an $i$-causal serialization of RedBlue order $O$. 
A Example of Bank

- Deposit
- Withdraw
- Accrue Interest

```plaintext
float balance, interest = 0.05;
func deposit( float money ):
    balance = balance + money;
func withdraw( float money ):
    if ( balance - money >= 0 ) then:
        balance = balance - money;
    else print "failure";
func accrueinterest():
    float delta = balance × interest;
    balance = balance + delta;
```
A Example of Bank

- Deposit
- Withdraw
- Accrue Interest

```
float balance, interest = 0.05;
func deposit( float money ):
    balance = balance + money;
func withdraw( float money ):
    if ( balance - money >= 0 ) then:
        balance = balance - money;
    else print "failure";
func accrueinterest():
    float delta = balance \times interest;
    balance = balance + delta;
```

Alice in EU
- balance: 100
  - deposit(20)
  - balance: 120
  - accrueinterest()
    - balance: 126

Bob in US
- balance: 100
  - accrueinterest()
    - balance: 105
  - deposit(20)
    - balance: 125

NOT commutable
Generator and Shadow Operation

- Split not commutable operation $u$ into:
  - Generator Operations $g_u$
    - Executed only at primary site against some state $S$ and produces shadow operations
    - $S + g_u = S$ (no side effect)
  - Shadow Operations $h_u(S)$
    - Executed at every site
    - $S + h_u(S) = S + u$
Generator and Shadow Operation (Cont’d)

• Generator make a decision $d$ based on a local state $S$
• Generator produces a colored shadow operation $h(d, S)$
• Shadow operation is applied at all site
Revisit the Example of Bank

```
float balance, interest = 0.05;
func deposit( float money ):
    balance = balance + money;
func withdraw( float money ):
    if ( balance - money >= 0 ) then:
        balance = balance - money;
    else print "failure";
func accrueinterest():
    float delta = balance × interest;
    balance = balance + delta;
```

```
func deposit’( float money ):
    balance = balance + money;
func withdrawAck’( float money ):
    balance = balance - money;
func withdrawFail’():
    /* no-op */
func accrueinterest’( float delta ):
    balance = balance + delta;
```

All operations are commutable
=> All can be blue!
Revisit the Example of Bank (Cont’d)

Alice in EU

100

Deposit'(20)

120

Deposit'(20)

125

Accrueinterest'(5)

Bob in US

100

Deposit(20)

Deposit'(20)

Accrueinterest'(5)

105

Accrueinterest'(5)

125

Deposit'(20)
Maintaining Invariants

• Bank balance cannot be negative!!!

- Any shadow operation that may result in violating invariants must be red

```
func deposit' (float money):
    balance = balance + money;

func withdrawAck' (float money):
    balance = balance - money;

func withdrawFail' ():
    /* no-op */

func accrueInterest' (float delta):
    balance = balance + delta;
```
What can be blue?
What must be red?

• For any pair of non-commutative shadow operations \( u \) and \( v \), label both \( u \) and \( v \) red.
• For any shadow operation \( u \) that may result in an invariant being violated, label \( u \) red.
• Label all non-red shadow operations blue.
Gemini Implementation

- Optimistic Concurrency Control
  - Maintaining order by logical clocks
    - $< < b_0, b_1, ..., b_{k-1} > , r >$
  - Token passing scheme for red operation total ordering
    - Only the site that holds the token can issue a red operation and increase $r$
- Failure Handling
  - Not implemented
Gemini Implementation (Cont’d)

Alice in EU

Bob in US

Coordinator

Storage Engine

Storage Engine

 Coordinator

 A1

19
Gemini Implementation (Cont’d)

Alice in EU

Coordinator

Storage Engine

Bob in US

Coordinator

Storage Engine

AI

AI
Gemini Implementation (Cont’d)

Alice in EU

Bob in US

Coordinator

Storage Engine

A2

A1

A1

A1
Gemini Implementation (Cont’d)

Alice in EU

Bob in US

Coordinator

Storage Engine

Coordinator

Storage Engine

A1

A2

A2

A1
Gemini Implementation (Cont’d)

Alice in EU

Coordinator

Storage Engine

A2

A1

Bob in US

Coordinator

Storage Engine

B1

A1
Gemini Implementation (Cont’d)

Alice in EU

Coordinator

Storage Engine

Bob in US

Coordinator

B1

Storage Engine

A2

A1

A2

A1
Gemini Implementation (Cont’d)

Alice in EU

Coordinator

Storage Engine

A1

A2

Bob in US

Coordinator

Storage Engine

B1

B2

A1

A2
Gemini Implementation (Cont’d)

Alice in EU

Coordinator

Storage Engine

Bob in US

Coordinator

Storage Engine

A1

A2

A3

B1

B2

B1

A2

A1
Gemini Implementation (Cont’d)

Alice in EU

Coordinator

Storage Engine

Bob in US

Coordinator

Storage Engine

A1

A2

A3

B1

B2

A1

B2

B1

A2

A3
Gemini Implementation (Cont’d)

Alice in EU

Bob in US

Coordinator

Storage Engine

Coordinator

Storage Engine
• Amazon EC2
  o US East (UE)
  o US West (UW)
  o Ireland (IE)
  o Brazil (BR)
  o Singapore (SG)

<table>
<thead>
<tr>
<th></th>
<th>UE</th>
<th>UW</th>
<th>IE</th>
<th>BR</th>
<th>SG</th>
</tr>
</thead>
<tbody>
<tr>
<td>UE</td>
<td>0.4 ms 994 Mbps</td>
<td>85 ms 164 Mbps</td>
<td>92 ms 242 Mbps</td>
<td>150 ms 53 Mbps</td>
<td>252 ms 86 Mbps</td>
</tr>
<tr>
<td>UW</td>
<td>0.3 ms 975 Mbps</td>
<td>155 ms 84 Mbps</td>
<td>207 ms 35 Mbps</td>
<td>181 ms 126 Mbps</td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td>0.4 ms 996 Mbps</td>
<td>235 ms 54 Mbps</td>
<td>350 ms 52 Mbps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>0.3 ms 993 Mbps</td>
<td>380 ms 65 Mbps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SG</td>
<td>0.3 ms 993 Mbps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Average round trip latency and bandwidth between Amazon sites.
Microbenchmark Experiment

- Each user issues requests accessing a random record from a MySQL database
  - Each request maps to a single shadow operation
  - 10 Tables
  - 1,000,000 records
    - 1 text, 4 integer
    - 1.0 GB in total
(a) Blue request latency for all users as number of sites increases

(b) Red request latency for all users as number of sites increases

(c) Blue latency CDF for Singapore users as number of sites increases

(d) Red latency CDF for Singapore users as number of sites increases
Microbenchmark Experiment (Cont’d)

Figure 9: Throughput versus latency graph for a 2 site configuration with varying red-blue workload mixes.
Case Studies

- **TPC-W**
  - Online book store
- **RUBiS**
  - Auction website
- **Quoddy**
  - Social network

<table>
<thead>
<tr>
<th>Application</th>
<th>user requests</th>
<th>transactions total</th>
<th>read-only</th>
<th>update</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-W</td>
<td>14</td>
<td>20</td>
<td>13</td>
<td>7</td>
<td>9k</td>
</tr>
<tr>
<td>RUBiS</td>
<td>26</td>
<td>16</td>
<td>11</td>
<td>5</td>
<td>9.4k</td>
</tr>
<tr>
<td>Quoddy</td>
<td>13</td>
<td>15</td>
<td>11</td>
<td>4</td>
<td>15.5k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>shadow operations</th>
<th>LOC changed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>blue no-op</td>
<td>blue update</td>
</tr>
<tr>
<td>TPC-W</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>RUBiS</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>Quoddy</td>
<td>11</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4: Proportion of blue and red shadow operations and read-only and update requests in TPC-W and RUBiS workloads at runtime.
TPC-W Results

(a) TPC-W doCart

(b) TPC-W doBuyConfirm

(a) TPC-W shopping mix
Conclusions

• RedBlue consistency allows strong consistency and eventual consistency to coexist
• Generator/shadow operations extend the space of fast operations
• A precise labeling methodology allows systems to be fast and behave as expected
• Experiment results show that both latency and throughput are improved