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

Authors: Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues

Presenter: Devesh Kumar Singh
GEO-REPLICATION

- Internet users globally distributed
- Higher Latency => poor user experience => loss in revenue
- Replicate data across geographically diverse sites
- Serve users from closest/least loaded site
- Need to decide on consistency models
• Single server behavior with natural semantics like linearizability
• Coordination overhead between replicas, amplified in geo-replication
• High Latency for remote clients
• e.g. Yahoo PNUTS
Multi server behavior with short term state divergence
- Conflict resolution by last writer wins etc.
- Low latency for remote users, might cause undesirable behavior
- e.g. Amazon Dynamo
### RED-BLUE CONSISTENCY

**Strong consistency**

- Totally ordered **Red** ops

**Red-Blue consistency**

- Low latency **blue** ops when possible
- Coordination for **Red** ops only when necessary

**Eventual Consistency**

- Partially ordered **Blue** ops
Red-Blue Consistency

- RedBlue order
  - Red operations must be totally ordered
  - Blue operations order can vary from site to site

Site A:  A1 B1 R1 B2 A2 R2 R3 B3
Site B:  B1 B2 A1 R1 R2 A2 B3 R3

- Causal serialization
  - A site has a causal serialization of the RedBlue order if the ordering is a linear extension of the RedBlue order

- State convergence
  - All causal serializations of the RedBlue order reach the same state
  - All Blue orders must be globally commutative

- Red Blue Consistency
  - Each site applies operations according to the causal serialization of the RedBlue order
Initial: balance = 100, interest = 0.05

Alice in EU

100

deposit(20)

120

accrueinterest()

Bob in US

100

accrueinterest()

105

deposit(20)

125


deposit(float money) {
    balance = balance + money;
}

withdraw(float money) {
    if (balance - money >= 0)
        balance = balance - money;
    else
        print "failure";
}

accrueinterest() {
    float delta = balance * interest;
    balance = balance + delta;
}
Problem: Different execution order lead to divergent state

Cause: accrueinterest doesn’t commute with deposit

Solution: Mark all as Red for convergence, but Red is slow

Better Solution: Split non-commutative operations into two
   - Compute the amount of interest accrued
   - Treat computed value as deposit

```python
accrueinterest():
   delta = balance * interest
   balance = balance + delta

accrueinterest_gen():
   delta = balance * interest
   accrueinterest(delta):
   balance = balance + delta
```
Generator/Shadow Operations

- **Generator Operation**
  - Only executed at the primary site against a system state
  - Produces no side effects
  - Determines state transitions that would occur
  - Produces shadow operations

- **Shadow Operation**
  - Applies the state transitions to all the sites including the primary site
  - Must produce the same effects as the original operation given the original state for the Generator operation
BANKING SYSTEM REVISITED

**Original/Generator operation**

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

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

withdraw(float m)
{
    if(balance - m >= 0)
        balance = balance - m;
    else
        print "Error"
}
```

**Shadow operation**

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

accrueinterest'(float delta)
{
    balance = balance + delta;
}

withdrawAck'(float m)
{
    balance = balance - m;
}

withdrawFail'()
{
}
```
FAST AND CONSISTENT BANK

Initial: balance = 100, interest = 0.05

Alice in EU

Initial: balance = 100, interest = 0.05

Bob in US

deposit(20): +20

accrueinterest: +5
ANOTHER ISSUE

Initial: balance = 100, interest = 0.05

Alice in EU
- 125
- 100
- 25
- 80
- 55

withdraw(100): -100

Bob in US
- 125
- 80
- 45
- 100
- 55

withdraw(80): -80
ANOTHER ISSUE

- Problem: Different execution orders lead to a negative balance.
- Cause: Blue operations that potentially break invariants execute without coordination.
- Solution: We must label successful withdrawal (withdrawAck’) as Red
(a) RedBlue order $O$ of banking shadow operations

(b) Convergent and invariant preserving causal serializations of $O$
EVALUATION

- Experiments with:
  - E-commerce benchmarks: TPC-W, RUBiS
  - Social networking app: 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
# Most Operations Are Blue

<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>
# Most Operations Are Blue

<table>
<thead>
<tr>
<th>Apps</th>
<th>workload</th>
<th>Originally</th>
<th></th>
<th>With shadow ops</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blue (%)</td>
<td>Red (%)</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>
<td>99.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Shopping mix</td>
<td>85.0</td>
<td>15.0</td>
<td>99.2</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Ordering mix</td>
<td>63.0</td>
<td>37.0</td>
<td>93.6</td>
<td>6.4</td>
</tr>
<tr>
<td>RUBiS</td>
<td>Bidding mix</td>
<td>85.0</td>
<td>15.0</td>
<td>97.4</td>
<td>2.6</td>
</tr>
<tr>
<td>Quoddy</td>
<td>a mix with 15% update</td>
<td>85.0</td>
<td>15.0</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>
IMPROVED USER OBSERVED LATENCY

Average latency for users at all five sites

Latency (ms)

US-East  US-West  Ireland  Brazil  Singapore

1-site original TPC-W

5-site TPC-W with Gemini
THROUGHPUT SCALES WITH NO OF SITES

Peak throughput for different deployments
SUMMARY

- 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.
THANK YOU