# Paper 1: Li et al.



**Title** Making Geo-Replicated Systems Fast as Possible,Consistent when necessary
*10th USENIX Symposium on Operating Systems Design and Implementation*
Authors
Date

# Motivation:

1. To improve user-experience, services replicate system state across geographical diverse sites.
2. Performance vs Consistency
   - Amazonś Dynamo - eventual consistency where state temporarily converge.
   - Yahoo PNUTS - avoids state divergence by requiring all operations that update the service state to be funneled through a primary site and thus incurring increased latency.

# Overview:

1. RedBlue Consistency - Blue operations execute locally and are lazily replicated.Red operations are serialized with respect to each other and are immediately cross-site coordinated.
2. Conditions under which operations must be colored red or blue.
3. Decomposing operations into two components - a generator operation and a shadow operation.

# Properties of Geo-Replicated Systems



1. Low latency - Operations should proceed after contacting a small number of users.
2. Causality - Monotonicity of user request within session and also preserving causality across clients
3. State Convergence - All replicas have executed the same set of operations
4. All operations should return a single value.
5. The system should provide a set of stable histories and support for general operations.
6. The system should preserve a set of invariants.
7. Eventual Propagation

# Related Work: Consistency



| Consistency level | Example systems | Immediate response | State convergence | Single value | General operations | Stable histories | Classification strategy |
|---|---|---|---|---|---|---|---|
| Strong | RSM [20, 31] | no | yes | yes | yes | yes | N/A |
| Timeline/snapshot | PNUTS [8], Megastore [3] | reads only | yes | yes | yes | yes | N/A |
| Fork | SUNDR [24] | all ops | no | yes | yes | yes | N/A |
| Eventual | Bayou [38], Depot [26] | all ops | yes | no | yes | yes | N/A |
| | Sporc [12], CRDT [33] | all ops | yes | yes | no | yes | N/A |
| | Zeno [34], COPS [25] | weak/all ops | yes | yes | yes | no | no / N/A |
| Multi | PSI [35] | cset | yes | yes | partial | yes | no |
| | lazy repl. [19], Horus [39] | immed./causal ops | yes | yes | yes | yes | no |
| RedBlue | Gemini | Blue ops | yes | yes | yes | yes | yes |

Table 1: Tradeoffs in geo-replicated systems and various consistency levels.

# Related Work: Levels of Consistency

1. Strong Consistency - Replicated systems behave like a single server that serialize all operations.
2. Timeline/Snapshot Consistency - THere is a total order for updates to the service state but gives the option of reading a consistent but dated view of the service.
3. Fork Consistency - Relaxes strong consistency by allowing users to observe distinct casual histories.
4. Eventual Consistency - All replicas "eventually" diverge at some state.
5. Multi Consistency - Other systems expose multiple values from divergent branches in operation replies either directly to the client or to an application-specific conflict resolution procedure.
6. RedBlue Consistency - Operations have multiple consistency levels

# Related Work: Other



1. Consistency Rationing - Consistency guarantees associated with the data instead of the operation.Also switches consistency levels at runtime.
2. TACT - bounds the amount of inconsisteny based on parameters like numeric errors,order errors, staleness etc.

# System Model - Assumptions

1. A distributed system with state fully replicated across $k$ sites denoted $site_0 \dots site_{k-1}$
2. $s \in S$ denotes a system state and $u, v \in O$ a set of operations.
3. Initial State - $S_0$. When operation $u$ is applied it goes to state $S'$. So $S' = S + u$
4. $\forall$ S $\in S$, S$+u + v =$ S $+v + u$
5. A state $S$ is valid if it satisfies all these invariants.
6. Each $u$ is submitted to one site which is called $u's$ primary site and denoted by $site(u)$.
7. The system later replicates $u$ to the other sites.

# RedBlue Consistency



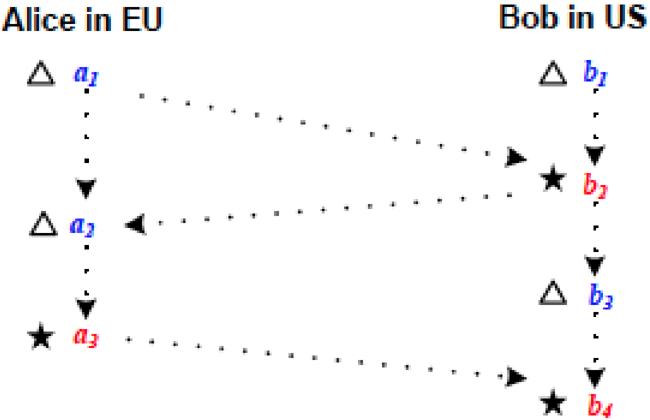
- RedBlue order : Given a set of operations $U = B \cup R$,where $B \cap R = \emptyset$, a RedBlue order is a partial order $O = (U, \prec)$ with the restriction that $\forall u, v \in R$ such that $u \neq v, u \prec v$ or $v \prec u$ ($i.e.$ red operations are totally ordered).
- Causal Serialization : Given a site $i$, $O_i = (U, <)$ is an $i$-causal serialization(or short, a causal serialization) of RedBlue order $O = (U, \prec)$ if
  1. $O_i$ is a linear extension of O ($i.e$, ¡ is a total order compatible with th partial order $\prec$ )
  2. for any two operations $u, v \in U$, if $site(v) = i$ and $u < v$ in $O_i$ then $u \prec v$

# RedBlue Consistency - Definition



- RedBlue consistency : A replicated sytem os O-RedBlue consistent(or short,RedBlue consistent) if each site i applies operations according to an i-causal serialization of RedBlue order $O$.

# Example




(a) RedBlue order $O$ of operations

# Example




(b) Causal serializations of $O$

# State Convergence



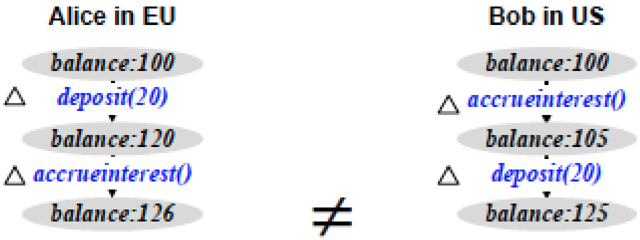
- A RedBlue consistent system is state convergent if all causal serializations of the underlying RedBlue order $O$ reach the same state $S$.

# State Convergence:Example

```
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;
```

# State Convergence:Example


(a) RedBlue order $O$ of operations issued by Alice and Bob



# State Convergence: Theorem

**Theorem**

*Given a RedBlue order O, if all blue operations are globally commutative then any O-RedBlue consistent system is state convergent*

# Replicating side effects -Generator Operation and Shadow Operation



1. Generator Operation $g_u$ - executed only at primary site against some system state $S$.
2. Shadow Operation $h_u(S)$ - executed at every site(including the primary site)

# Replicating side effects - Defining shadow operations

1. Correct Generator/ Shadow Operations : The decomposition of operation $u$ into generator and shadow operations is correct if for all states $S$, the generator operation $g_u$ has no effect and the generated shadow operation $h_u(S)$ has the same effect as $u$, *i.e.*, for any state $S : S + g_u = S$ and $S + h_u(S) = S + u$

# Replicating side effects -Revisiting RedBlue consistency



1. Given a site $i$, $O_i = (U \cup V_i, <)$ is an $i$-causal serialization of RedBlue order $O = (U, \prec)$ if
   - $O_i$ is a total order;
   - $(U, <)$ is a linear extension of $O$;
   - For any $h_v(S) \in U$ generated by $g_v \in V_i$, $S$ is the state obtained after applying the sequence of shadow operations preceeding $g_v$ in $O_i$;
   - For any $g_v \in V_i$ and $h_u(S) \in U$, $h_u(S) < g_v$ in $O_i$ iff $h_u(S) \prec h_v(S')$ in $O$.

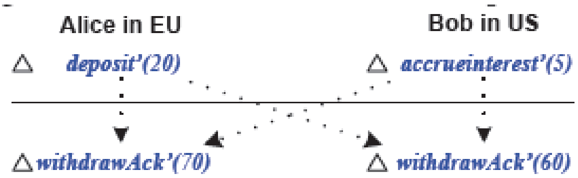
# Shadow Banking and Invariants - Example



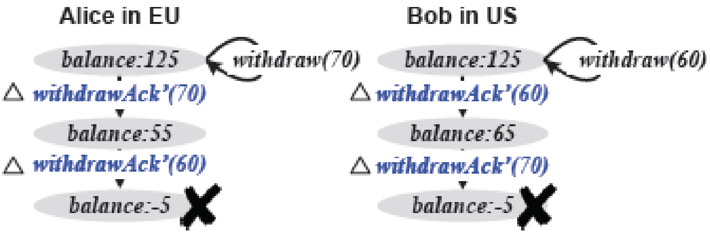
```
func deposit' ( float money ):
   balance = balance + money;
func withdrawAck' ( float money ):
   balance = balance - money;
func withdrawFail' ():
   /* no-op */
func accrueinterest' ( float delta ):
   balance = balance + delta;
```

# Shadow Banking and Invariants - Example






(a) RedBlue order $O$ of banking shadow operations



# Shadow Banking and Invariants - Example



- Invariant Safe - Shadow operation $h_u(S)$ is invariant safe if for all valid states $S$ and $S'$, the state $S' + h_u(S)$ is also valid.

**Theorem**

*If all shadow operations are correct and all blue shadow operations are invariant safe and globally commutative,then for any execution of that system that is RedBlue consistent, no site is ever in an invalid state.*

# What can be blue? What can be red?

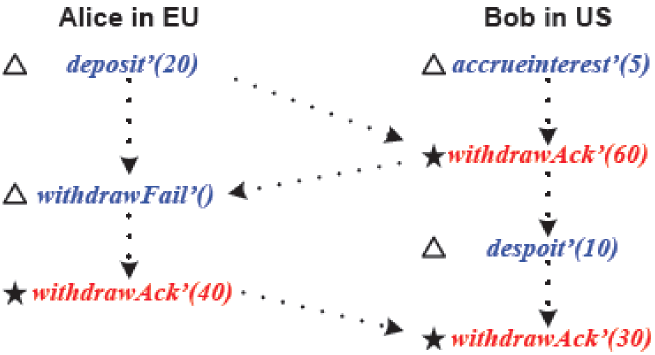


The procedure for deciding which shadow operations can be blue or must be red if a RedBlue consistent system is to provide both state convergence and invariant preservation:

1. For any pair of non-commutative shadow operations $u$ and $v$, label both $u$ and $v$ red.
2. For any shadow operation $u$ that may result in an invariant being violated, label $u$ red.
3. Label all non-red shadow operations blue.

# Shadow Banking and Invariants - Example

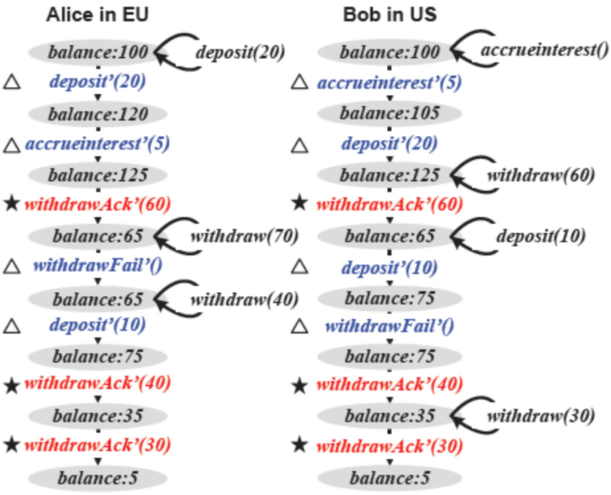





(a) RedBlue order $O$ of banking shadow operations

# Shadow Banking and Invariants - Example






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

# Gemini Design and Implementation - Prototype



1. It consisted of 10K lines of java code and uses MySQL as its storage back-end
2. Each Gemini site consists of :
   - a storage engine
   - a proxy server
   - a concurrency coordinator
   - a data writer
3. The single site is replicated across multiple sites.

# Gemini Design and Implementation - Basic Flow



1. User issues request to a proxy server located at the closest site.
2. The proxy server processes the request by executing an appropriate application transaction which is implemented as a single Gemini operation.
3. Storage Engine - Relational Database
4. Scratchpad Operations - Temporary tables

# Gemini Design and Implementation - Basic Flow



1. User issues request to a proxy server located at the closest site.
2. The proxy server processes the request by executing an appropriate application transaction which is implemented as a single Gemini operation.
3. Storage Engine - Relational Database
4. Scratchpad Operations - Temporary tables

# Gemini Design and Implementation - Failure Handling



1. Isolated Component Failure - Standard state machine replication techniques can be employes to make each component robust.
2. Site Failure - A fault tolerance consensus protocol like Paxos can be used.
3. Operation Propagation - This can be addressed by using standard techniques for exchaning causal logs or reliable multicast.
4. Cross-session monotonicity - This can be addressed by allowing the user to specify a "last read" version when starting a new session or requiring the user to cache all relevant requests in order to reply them when connecting to a new site.

# Case Studies

- TPC-W shopping cart benchmark
- RUBiS auction benchmark
- Quoddy social networking application

Two main tasks :

- Decomplosing the application into a generator and shadow operation
- Labeling the shadow operations appropriately

# Original application to RedBlue Consistent



| Application | Original | | | | | RedBlue consistent extension | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | user requests | transactions | | | LOC | shadow operations | | | | LOC changed |
| | | total | read-only | update | | blue no-op | blue update | red | LOC | |
| TPC-W | 14 | 20 | 13 | 7 | 9k | 13 | 14 | 2 | 2.8k | 429 |
| RUBiS | 26 | 16 | 11 | 5 | 9.4k | 11 | 7 | 2 | 1k | 180 |
| Quoddy | 13 | 15 | 11 | 4 | 15.5k | 11 | 4 | 0 | 495 | 251 |

Table 2: Original applications and the changes needed to make them RedBlue consistent.

# TPC-W



- Serves 14 diferent user requests such as browsing, searching, adding products to a shopping cart or placing an order.
- Each user request generates one to four transactions that access state stored across eight different tables.
- Shopping cart can be shared by multiple users across multiple sessions.

# TPC-W - Writing TPC-W generator and shadow operations



```
doBuyConfirm(cartId) {
 beginTxn();
 cart = exec(SELECT * FROM cartTb WHERE cId=cartId);
 cost = computeCost(cart);
 orderId = getUniqueId();
 exec(INSERT INTO orderTb VALUES(orderId, cart.item.id, cart.item.qty
        ,cost));
 item =exec(SELECT * FROM itemTb WHERE id=cart.item.id);
 if item.stock - cart.item.qty < 10 then:
    delta = item.stock - cart.item.qty + 21;
    if delta > 0 then:
       exec(UPDATE itemTb SET item.stock+ = delta);
    else rollback();
 else exec(UPDATE itemTb SET item.stock- = cart.item.qty);
 exec(DELETE FROM cartContentTb WHERE cId=cartId AND id=
        cart.item.id);
 commit();}
```

(a) Original transaction that commits changes to database.

# TPC-W - Writing TPC-W generator and shadow operations



```
doBuyConfirmGenerator(cartId){
 sp = getScratchpad();
 sp.beginTxn();
 cart = sp.exec(SELECT * FROM cartTb WHERE cId=cartId);
 cost = computeCost(cart);
 orderId = getUniqueId();
 sp.exec(INSERT INTO orderTb VALUES (orderId,cart.item.id,
        cart.item.qty,cost));
 item = sp.exec(SELECT * FROM itemTb WHERE id=cart.item.id);
 if item.stock - cart.item.qty < 10 then:
    delta = item.stock - cart.item.qty + 21;
    if delta > 0 sp.exec(UPDATE itemTb SET item.stock+ = delta);
    else sp.discard(); return;
 else sp.exec(UPDATE itemTb SET item.stock- = cart.item.qty);
 sp.exec(DELETE FROM cartTb WHERE cId=cartId AND id=cart.item.id);
 L_TS = getCommitOrder();
 sp.discard();
 if replenished return (doBuyConfirmIncre'(orderId,cartId,
        cart.item.id,cart.item.qty,cost,delta,L_TS));
 else return (doBuyConfirmDecre'(orderId,cartId,cart.item.Id,
        cart.item.qty,cost,L_TS));}
```

(b) Generator operation that manipulates data via a private *scratchpad*.

# TPC-W - Writing TPC-W generator and shadow operations

```
doBuyConfirmIncre'(orderId, cartId, iId, qty, cost, delta, L_TS){
 exec(INSERT INTO orderTb VALUES(orderId, iId, qty, cost, L_TS));
 exec(UPDATE itemTb SET item.stock += delta);
 exec(UPDATE itemTb SET item.l_ts = L_TS WHERE item.l_ts < L_TS);
 exec(UPDATE cartContentTb SET flag = TRUE WHERE id = iId AND
        cid = cartId AND l_ts <= L_TS);}
```

(c) Shadow doBuyConfirmIncre' (Blue) that replenishes the stock value.

```
doBuyConfirmDecre'(orderId, cartId, iId, qty, cost, L_TS){
 exec(INSERT INTO orderTb VALUES(orderId, iId, qty, cost, L_TS));
 exec(UPDATE itemTb SET item.stock -= qty);
 exec(UPDATE itemTb SET item.l_ts = L_TS WHERE item.l_ts < L_TS);
 exec(UPDATE cartContentTb SET flag = TRUE WHERE id = iId AND
        cid = cartId AND l_ts <= L_TS);}
```

(d) Shadow doBuyConfirmDecre' (Red) that decrements the stock value.

# Evaluation-Experimental Setup



- Experiments were run on Amazon EC2 using 5 virtual machine instances located in 5 sites - US east(UE), US west(UW), Ireland(IE), Brazil(BR) and Singapore(SG).
- Each VM has 8 virtual cores and 15 GB of RAM. VMs run Debian 6( Squeeze) 64 bit, MYSQL 5.5.18,Tomcat 6.0.35 and Sun Java SDK 1.6.

# Evaluation-Experimental Setup



| | UE | UW | IE | BR | SG |
|---|---|---|---|---|---|
| UE | 0.4 ms<br>994 Mbps | 85 ms<br>164 Mbps | 92 ms<br>242 Mbps | 150 ms<br>53 Mbps | 252 ms<br>86 Mbps |
| UW | | 0.3 ms<br>975 Mbps | 155 ms<br>84 Mbps | 207 ms<br>35 Mbps | 181 ms<br>126 Mbps |
| IE | | | 0.4 ms<br>996 Mbps | 235 ms<br>54 Mbps | 350 ms<br>52 Mbps |
| BR | | | | 0.3 ms<br>993 Mbps | 380 ms<br>65 Mbps |
| SG | | | | | 0.3 ms<br>993 Mbps |

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

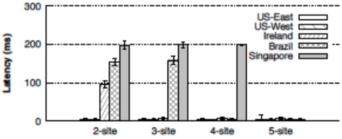
# Evaluation-Microbenchmark



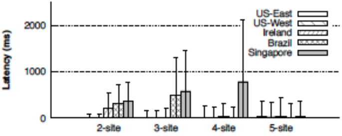
- Each user issues requests accessing a random record from a MySQL database.
- Each request maps to a single shadow operation
- A request is blue if it maps to a blue shadow operation and red otherwise
- Dataset consists of 10 tables,each initialized with 1000000 records,each record has 1 text and 4 integer attributes. The total size of the dataset is 1.0 GB.

# Evaluation-Microbenchmark

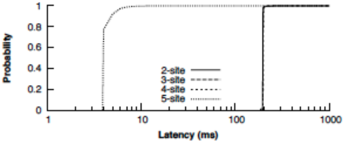


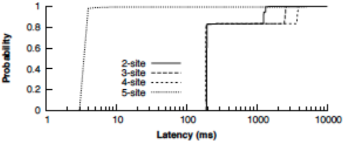



(a) Blue request latency for all users as number of sites increases



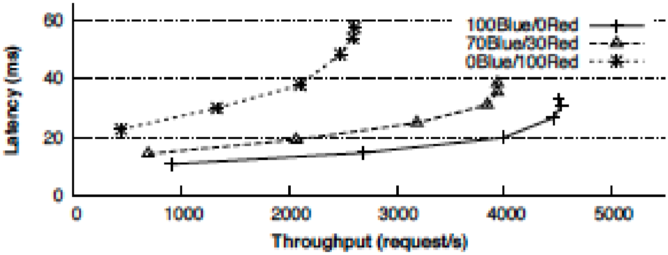

(b) Red request latency for all users as number of sites increases



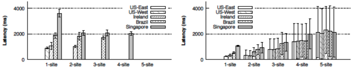



# Evaluation-Peak Throughput

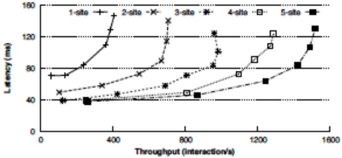




# Evaluation-TPC-W



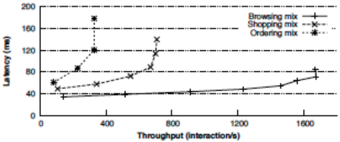

(a) TPC-W doCart (b) TPC-W doBuyConfirm




(a) TPC-W shopping mix

# Conclusion

# Evaluation-TPC-W




| | TPC-W shopping | |
|---|---|---|
| | Original | Gemini |
| Thput. (inter/s) | 409 | 386 |
| Avg. latency | 14 ms | 15 ms |

# Paper 2: Lloyd et al.



**Stronger Semantics for Low-Latency Geo-Replicated Storage**
*Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI13)*
Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen
April 2013

# Main Idea

# Main Idea



- Take slight hit in throughput to get stronger version of consistency

# Main Idea



- Take slight hit in throughput to get stronger version of consistency
- Causal Consistency Instead of Eventual Consistency (causal is stronger)

# Main Idea

- Take slight hit in throughput to get stronger version of consistency
- Causal Consistency Instead of Eventual Consistency (causal is stronger)
- We require low latency

# Main Idea



- Take slight hit in throughput to get stronger version of consistency
- Causal Consistency Instead of Eventual Consistency (causal is stronger)
- We require low latency
- Extend previous systems: Cassandra and COPS

# Contributions

# Contributions



- Eiger
  - Low Latency
  - High throughput (slightly lower than Cassandra)
  - Causal Consistency (rather than eventual as in Cassandra)

# Contributions



- Eiger
  - Low Latency
  - High throughput (slightly lower than Cassandra)
  - Causal Consistency (rather than eventual as in Cassandra)
- Read Only Algorithm

# Contributions



- Eiger
  - Low Latency
  - High throughput (slightly lower than Cassandra)
  - Causal Consistency (rather than eventual as in Cassandra)
- Read Only Algorithm
- Write Only Algorithm

# Background

# Background



- Cassandra
  - Eventual Consistency

# Background



- Cassandra
  - Eventual Consistency
- COPS

# Consistency - Causal versus Eventual

# Consistency - Causal versus Eventual

- p1

# Column Family Data Model

# Column Family Data Model



- p1

# Column Family Data Model



- p1
- p2

# Column Family Data Model



- p1
- p2
- p3

# Eiger

# Eiger



- p1

# Eiger

- p1
- p2

# Eiger



- p1
- p2
- p3

# Evaluation

# Evaluation

- Versus Cassandra
  - Within 7% of throughput Using Facebook-like data
    - Ops/sec
    - Keys/sec
    - Columns/sec

# Evaluation



- Versus Cassandra
  - Within 7% of throughput Using Facebook-like data
    - Ops/sec
    - Keys/sec
    - Columns/sec
- Versus COPS

# Follow Up Research

# Follow Up Research



- p1

# Ideas for Future Research

# Ideas for Future Research



- p1

# Bibliography




- **Stronger Semantics for Low-Latency Geo-Replicated Storage**, *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI13)*, Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen, April 2013