Achieving Serializability with Chains

Chain Property
1. All-or-nothing atomicity
   If the first hop (T_{1,1}) commits, the rest of the chain (T_{1,2} and T_{1,3}) will eventually commit.

2. Origin ordering
   If chains T_1 and T_2 start on the same server, and T_1 executes before T_2, then T_1 executes before T_2 at every server where they both execute.

Problem
Interleaved execution of chains might not be serializable.

Static Analysis
4(6inge)-edge: between hops in the same chain
C(onflicting)-edge: between R/W, W/W hop pairs

No SC-cycle = Serializability

Key Techniques to Avoid SC-Cycles
1. Programmers can annotate which pair of hops commute
   ✓ No C-edge between two commutative hops
2. Origin ordering to remove C-edges in system chains

Key Ideas of Lynx
1. Decompose transactions to sequence of steps on multiple shards
   ✓ Hops execute on separate servers independently
   ✓ Allow fast return from first hop (local datacenter, low latency)
   ✗ Unsafe (might violate serializability)

Observation: web applications use a predefined set of transaction steps
2. Static analysis to determine safe and efficient chain execution
   ✓ Guarantee serializability

How to Use Chains

1. System Chains
   ✓ Secondary Index
   ✓ Join View
   ✓ Geo-replication

2. User Defined Chains
   Programmers write chains in JavaScript

Prototype Performance

<table>
<thead>
<tr>
<th></th>
<th>QPS (K/s)</th>
<th>Throughput</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lynx</td>
<td>184</td>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>Cassandra</td>
<td>173</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>Follow</td>
<td>1350</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>Tweet</td>
<td>2770</td>
<td>27</td>
<td>113</td>
</tr>
<tr>
<td>Timeline</td>
<td>42</td>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>

Motivation
Web applications rely on geo-distributed storage systems as their backend

Currently, there is a hard choice between:
1. Serializable transactions with high latency
2. Limited transactions with relaxed semantics

Low latency (intradatacenter)  High latency (interdatacenter)

Our goal, get best of both worlds:
1. Achieve serializability with low latency
2. Provide easy-to-use relational model

Web App Storage

Transaction Chains: Achieving Serializability with Low Latency in Geo-Distributed Storage Systems
Yang Zhang * Russell Power * Siyuan Zhou * Yair Sovran * Marcos K. Aguilera ‡ Jinyang Li *
* New York University  ‡ Microsoft Research Silicon Valley


Motivation

Web applications rely on geo-distributed storage systems as their backend

Currently, there is a hard choice between:
1. Serializable transactions with high latency
2. Limited transactions with relaxed semantics

Our goal, get best of both worlds:
1. Achieve serializability with low latency
2. Provide easy-to-use relational model

Key Ideas of Lynx
1. Decompose transactions to sequence of steps on multiple shards
   ✓ Hops execute on separate servers independently (no distributed locking, cheap synchronization operation)
   ✓ Allow fast return from first hop (local datacenter, low latency)
   ✗ Unsafe (might violate serializability)

Observation: web applications use a predefined set of transaction steps
2. Static analysis to determine safe and efficient chain execution
   ✓ Guarantee serializability

How to Use Chains

1. System Chains
   ✓ Secondary Index
   ✓ Join View
   ✓ Geo-replication

2. User Defined Chains
   Programmers write chains in JavaScript

Prototype Performance

<table>
<thead>
<tr>
<th></th>
<th>QPS (K/s)</th>
<th>Throughput</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lynx</td>
<td>184</td>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>Cassandra</td>
<td>173</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>Follow</td>
<td>1350</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>Tweet</td>
<td>2770</td>
<td>27</td>
<td>113</td>
</tr>
<tr>
<td>Timeline</td>
<td>42</td>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>

Motivation
Web applications rely on geo-distributed storage systems as their backend

Currently, there is a hard choice between:
1. Serializable transactions with high latency
2. Limited transactions with relaxed semantics

Low latency (intradatacenter)  High latency (interdatacenter)

Our goal, get best of both worlds:
1. Achieve serializability with low latency
2. Provide easy-to-use relational model

Key Ideas of Lynx
1. Decompose transactions to sequence of steps on multiple shards
   ✓ Hops execute on separate servers independently (no distributed locking, cheap synchronization operation)
   ✓ Allow fast return from first hop (local datacenter, low latency)
   ✗ Unsafe (might violate serializability)

Observation: web applications use a predefined set of transaction steps
2. Static analysis to determine safe and efficient chain execution
   ✓ Guarantee serializability

How to Use Chains

1. System Chains
   ✓ Secondary Index
   ✓ Join View
   ✓ Geo-replication

2. User Defined Chains
   Programmers write chains in JavaScript

Prototype Performance

<table>
<thead>
<tr>
<th></th>
<th>QPS (K/s)</th>
<th>Throughput</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lynx</td>
<td>184</td>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>Cassandra</td>
<td>173</td>
<td>2</td>
<td>36</td>
</tr>
<tr>
<td>Follow</td>
<td>1350</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>Tweet</td>
<td>2770</td>
<td>27</td>
<td>113</td>
</tr>
<tr>
<td>Timeline</td>
<td>42</td>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>