Rococo: Extract more concurrency from distributed transactions

Shuai Mu, Yang Cui, Yang Zhang, Wyatt Lloyd, Jinyang Li
Tsinghua University, New York University, University of Southern California, Facebook
What Large Web Sites Need

36.8 million sold/day
170 million sold/day
$169 billion trade/day

Scalable Storage w/ Transactions!
What is a Distributed Transaction?

It is not shown in the image, but the code snippet below illustrates a distributed transaction:

```
BEGIN_TX
if (iphone > 0) {
    iphone--;
}
if (case > 0) {
    case--;
}
END_TX
```

Transactions should be strictly serializable! Otherwise…
Loss of serializability = angry customer
Serializability is Costly under Contention

Throughput transaction/s

# of concurrent transactions

- Two-Phase Locking (2PL)
- Optimistic Concurrency Control (OCC)
OCC Aborts Contended Transactions

```java
if (iphone > 0) {
    iphone--;
}
if (case > 0) {
    case--;
}
```

Two-Phase Commit (2PC)

```java
if (iphone > 0) {
    iphone--;
}
if (case > 0) {
    case--;
}
```
If (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
Achieve serializability w/o aborting or blocking*  

* for common workloads
Rococo’s Approach

Defer piece execution to enable reordering

```c
if (iphone > 0) {
    iphone--;
}
if (case > 0) {
    case--;
}
```

```c
if (iphone > 0) {
    iphone--;
}
if (case > 0) {
    case--;
}
```

```c
if (iphone > 0) {
    iphone--;
}
if (case > 0) {
    case--;
}
```
Rococo Overview: Key techniques

1. Two-phase protocol
   - **Most** pieces are executed at the second phase.

2. Decentralized dependency tracking
   - Servers track pieces’ arrival order
   - Identify non-serializable orders
   - Deterministically reorder pieces

3. Offline workload checking
   - Identifies safe workloads (common)
   - Identifies small parts that need traditional approaches (rare)
#1 Two-phase protocol

**Start Phase**
- Send pieces to servers w/o executing them
- Set up a provisional order

**Commit Phase**
- Establish a final order and execute pieces
- Reorder for serializability
#2 Dependency Tracking: Start Phase

T1:
- if ... iphone--
- if ... case--

T2:
- if ... iphone--
- if ... case--

T1:
- if ... iphone--

T2:
- if ... case--

Case=1

iPhone=1
#2 Dependency Tracking: Commit Phase

Sort the cycle by any deterministic order
Problem: Not Every Piece is Deferrable

oid = next_oid ++
if ... iphone--
if ... case--
orderline.insert(oid, …)

Intermediate Results Calls for Immediate Execution

next_oid

oid = next_oid ++
if ... iphone--

iPhone

Case

oid

orderline

orderline.insert(oid, …… )
Immediate Pieces are Naughty!

1. Common workloads have few immediate pieces
2. Pre-known workload $\rightarrow$ Offline check

```c
a = next_a++;  
b = next_b++;  
ol_a.insert(a, ...);  
ol_b.insert(b, ...);
```

```c
a = next_a++;  
b = next_b++;  
ol_a.insert(a, ...);  
ol_b.insert(b, ...);
```
#3: Offline Checking: Basic

SC-cycles represent potential non-serializable executions that require 2PL/OCC
#3: Offline Checking: Enhanced

SC-cycles with deferrable pieces can be safely reordered!
Incorporating Immediate Pieces

Every cycle contains deferrable pieces, ensured by offline checking

A cycle with deferrable pieces is safe to reorder
Deferred Execution
Decentralized Dependency Tracking
Offline Checking

Merged Pieces
Read-only Transactions
Reducing Dep. Size
Fault Tolerance
Overlapping Trans.
Evaluation

• How does Rococo perform under contention?
• How does Rococo scale?
Workload: Scaled TPC-C

- One warehouse, many districts
- Partitioned by districts - all transactions distributed

New order (45%)

Payment (43%)

Delivery (4%)

Read-only: OrderStatus (4%), StockLevel (4%)
Rococo Has Higher Throughput

Throughput (new-order/s)

Concurrent reqs/server

Rococo
2PL
OCC

Increasing graph size

Aborts due to deadlock detection
Aborts due to conflicts in validation

8 servers, 10 districts per server → Main source of contention is next_oid of each district
Rococo Avoids Aborts

![Graph showing commit rate vs. concurrent requests per server for Rococo, 2PL, and OCC. The graph illustrates that Rococo has a lower commit rate at higher concurrency, with 61-66ms, whereas 2PL has a commit rate of 152-392ms, and OCC has a commit rate of 173-645ms.]
Rococo Scales Out

Throughput (new-order/s)

- Rococo
- 2PL
- OCC

Almost Linear
Blocking
Aborts

10 districts per server, fixed # of items → contention grows slowly
<table>
<thead>
<tr>
<th>System</th>
<th>Isolation Level</th>
<th>Concurrency Control Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rococo</td>
<td>Strict-Serial.</td>
<td>Rococo</td>
</tr>
<tr>
<td>Calvin [SIGMOD’12]</td>
<td>Strict-Serial.</td>
<td>Pre-ordered Locking</td>
</tr>
<tr>
<td>HStore [VLDB’07]</td>
<td>Strict-Serial.</td>
<td></td>
</tr>
<tr>
<td>Granola [SIGMOD’10]</td>
<td>S.I.</td>
<td></td>
</tr>
<tr>
<td>COPS [SOSP’11]</td>
<td>Causal</td>
<td>Dependency Tracking</td>
</tr>
</tbody>
</table>

Decentralized dependency tracking

Centralized sequencing layer, difficult to scale
Conclusion

- Traditional protocols perform poorly w/ contention
  - OCC aborts & 2PL blocks

- Rococo defers execution to enable reordering
  - Strict serializability w/o aborting or blocking for common workloads

- Rococo outperforms 2PL & OCC
  - With growing contention
  - Scales out
Thank you!

Questions?

https://github.com/msmummy/rococo

Poster tonight!