FlurryDB
A Dynamically Scalable Relational Database with Virtual Machine Cloning

Michael Mior and Eyal de Lara
University of Toronto
Cluster Databases

- We use read-one write all (ROWA) replication
  - Send reads to any server instance
  - Send writes to all server instances
- Two-phase commit is required for writes
Problem

- Loads may fluctuate drastically over time
- Pay-as-you-go IaaS clouds should be ideal

However

We want to adapt the size of the cluster to match the load
Problem

- Databases have large state and are hard to scale
- A complete copy may be required for new instances
- Copying large databases may take hours

Queries

Replication

copy data  boot replica  catch up  add replica

- Overprovisioning is necessary to maintain service
Solution – Virtual Machine Fork

- Analogous to fork() for OS processes
- Clones start immediately
- State fetched on-demand
  - Page faults fetch the memory or disk page

This can reduce instantiation time from minutes or hours to seconds
VM Fork

Master VM
FlurryDB

- Use VM fork to provision new instances and add elasticity to unmodified MySQL

- Making distributed commit cloning-aware to handle in-flight transactions
FlurryDB Challenges

1. Incorporate new worker into cluster

2. Preserve application semantics
Incorporate new worker

- Clone must connect to the load balancer
- Load balancer must begin sending transactions to the clone
Preserve application semantics

- Transactions may be in-progress at the time of cloning
- Clone gets new IP address
- Doing nothing drops connections and transaction status is unknown
Solutions to consistency

1. Employ a write barrier

2. Roll back all transactions in progress

3. Allow all transactions to complete
FlurryDB: Consistency beyond VM fork

Solution

Use a proxy which is aware of VM fork inside the virtual machine to maintain the database connection
Two-phase commit during cloning
Replica addition delay

- Update 10,000 rows concurrently

- Clone and measure replica addition delay in two cases
  - **Write barrier** - wait for outstanding writes to complete before cloning
  - **FlurryDB** - use double-proxying to allow completion on the clone
Replica addition delay

![Graph showing replica addition delay with Concurrent connections on the x-axis and Replica addition time (s) on the y-axis. The graph compares 'Write barrier' and 'FlurryDB' with error bars indicating variability.](image-url)
Proxy overhead

• Measurement of large SELECT transfer times shows ~5% drop in bandwidth

• Reconnection to new servers ~10x faster with no authentication
RUBBoS

- Simulates a news website such as Slashdot
- Users read, post, and comment on stories

- We run server at full capacity (25 clients)
- After 5 minutes, we clone to two servers and double the number of clients
- Throughput in queries per second is measured at the load balancer
RUBBOS Results

cloning at 300s

Graph showing Queries/s (thousands) vs. Time (s) with a vertical line at 300s indicating a change in behavior.
Summary

• FlurryDB adds elasticity to unmodified MySQL by interposing a cloning-aware proxy

• VM fork handles most of the issues with replica addition

• A proxy handles in-flight requests

• New replicas can be made available in seconds while maintaining consistency
Future Work

• Experiment with varying consistency protocols (e.g. master-slave, eventual consistency)

• Test scalability with larger numbers of clones

• Optimize virtual disk performance

• Provide support for transactional workloads
Questions?
Related Work

• “NoSQL” (perhaps more accurately, NoACID) systems using eventual consistency or key-value data models such as Cassandra or Dynamo
• Relational Cloud uses dynamic partitioning across a cluster of MySQL/PostgreSQL instances
• Urgaonkar et al. use rapid changes in resource allocate to provision a multi-tier Internet application
• Soundararajan et al. present dynamic replication policies for scaling database servers
• HyPer uses process fork and memory snapshots to enable a hybrid OLAP/OLTP workload