Intoducing Falcon

MySQL Users Conference 2006

Jim Starkey
(jstarkey@mysql.com)

First, what Falcon is

- A Transactional MySQL Storage Engine
- Based on Netfrastructure Database Engine
- Deployed in mission critical applications > 4 years
- Extended and integrated into MySQL environment

Second, what Falcon is **not**

- An InnoDB Clone
- Firebird
- A Firebird Clone
- A Standalone Database Management System
- Netfrastructure

Changes since the dawn of the RDMS (early 1980s)

- Uni-processors to SMP to multi-core SMP
- CPU performance: 1.7 to 4,200 MIPS
- Address space: 32 bits to 64 bits
- Memory speed: 1 microsecond to 7 nanoseconds
- Medium system memory: 2 MB to 1 GB
- Large system memory: 16 MB to 4+ GB
- Typical disk access time: 20 ms to 7 ms
- 100 users as “big” to 10,000 users as “small”

More Changes since the dawn of RDMS

- Decision Support => OLTP => Data Mining => Web
- Change in skill levels
  - DBAs are smarter and harder to find
  - Application programmers skill levels are down
  - Expert design now means appearance, not architecture

More Changes since the dawn of RDMS

Change in applications

- Much larger databases (delete is a forgotten verb)
- Many more queries per human interaction (esp. web)
- Few rows per result set
- Latency is more critical
- Blobs are much larger (and more important)
- Free context search!!!

What I have learned

- CPUs and memory are vastly faster
- Disks, relative to CPU and memory, are vastly slower
- MVCC works
- Record versions on disk are problematic
- Web applications are better applications and are the future
- People have more important things to do than tune databases

Falcon: The Engine Design for the Next 20 Years

Goal: Exploit large memory for more than just a bigger cache

Goal: Use threads and processors for data migration

Goal: Design to eliminate tradeoffs, minimize tuning

Goal: Scale gracefully to very heavy loads

Falcon Architectural Overview

- Basic model: Incomplete in-memory database with backfill from disk
- Two caches:
  - Traditional LRU page cache for disk
  - Larger row cache with age group scavenging
- Serial log for single write group commits

Falcon Architectural Overview

- Multi-version in memory, single version on disk
- All transaction state in memory with automatic overflow to disk
- Data and Indexes are single file plus log files
- Future: Blob repositories
- Future: Multiple page spaces

Falcon Implementation: Concurrency Control

- Basic model is multi-generational concurrency control (MVCC)
- Will be extended for relaxed consistency (but we do not approve!)
- Will be extended for a serializable mode

Falcon Implementation: Indexes

- Btree index with prefix compression
- No data (except key) in index
- Two-stage index retrievals
  - Index scan generates row bitmap
  - Records are fetched from disk in physical row order
- No performance difference between primary, secondary indexes
- Scan bitmaps can be combined for multi-index retrieval

Falcon Implementation: Data Flow

- Uncommitted row data is staged in memory (may overflow to scratch file). Indexes are updated immediately.
- On commit, row data is copied to the serial log and written
- Post commit, dedicated thread copies row data from serial log to data pages
- Page cache is periodically flushed to disk
- Blob data is queued for write at blob creation, backed out on rollback

Falcon Implementation: Data Reliability

- Physical structure protected by “careful write”
  - Database on disk is always valid, always consistent
  - Simple rule: A page is written before a pointer to it
- Atomicity protected by serial log
  - A transaction is committed when the commit record hits the oxide

Falcon Implementation: Data Reliability

Serial log is:

- A “do” log for post commit data migration
- A “redo” log for post-crash recovery of data
- An “undo” log for post-crash resource recovery

My Secret Agenda

- Replace varchar with “string” (forget the length)
- Replace tiny, small, medium, and big integers with “number”
- Adopt a security model useful for application servers
- Introduce useable row level security (filter sets)
- Teach the database world the merits of free context search (everyone else already knows)

Questions?