



PERCONA  
Performance Consulting Experts

---

# Scaling Applications with Caching Sharding and Replication

Apr 12, 2009

O'Reilly MySQL Conference  
and Expo

Santa Clara, CA

by Peter Zaitsev, Percona Inc

# Getting Started

- Look into Web Application types and their problems
  - Performance, Scalability, High Availability, Efficiency
- Learn how to solve them by
  - Caching and Buffering
  - Replication
  - Functional Partitioning and Sharding
- Explain which solutions work best in which cases

# Question Policy

- Ask your questions as I'm going through presentation.
- Hold off with longer questions to the end
- Do not hesitate to talk to me during conference
- Followup by email [pz@percona.com](mailto:pz@percona.com)

# Web Application Challenges

- Page Generation Layer
  - Scale by adding more servers
  - Most applications do not have interdependences
- Storage Layer (Static Content)
  - Images, Videos etc
  - No dependencies - scaling by more hard drives/boxes
  - CDN can often take the load
- “Database”
  - Often Hardest to scale due to complex interdependencies

# Classes of Web Applications

- New feature for existing service
  - Product recommendation on Amazon.Com
  - “Instant” high load and large database size
- Typical Startups
  - Slow but accelerating growth
  - Often have some time to fix problems
- Instant Hits
  - Ie some FaceBook Applications
  - Load Skyrockets within Days, Database size may follow

# Application Design Approaches

- “Think about today Style”
  - Make it work today and we'll see about tomorrow
  - Deliberate choice for speed of development or lack of skill
  - Typical for college startups
- “Best Practices Delivered”
  - Plan for Scaling, HA, Quality in advance
  - Do not sacrifice scaling even if it means longer time to deliver
  - Typical for established companies and second startups
- A lot of Applications are in the middle

# What is Sensible approach ?

- Define time horizon for which current architecture should live
  - “I'll build prototype, get funding in 3 months and hire smart guys to architect things right for me”
- Estimate performance requirements (load, database size etc). Better overestimate
- Plan your architecture to deliver these goals
  - Not scalable architecture can kill your app
  - Overkill in scalability can be too expensive and you may never get the product to the market.

# Example: E-Commerce site

- Over In development
- \$2M+ Development budget
- Well known Data Volume
- High investment in Advertisements
- Formal SLA with Partners on performance and availability
- Life time for years with limited changes
- Such systems tend to have more careful architecture design, capacity planning, load testing.

# Example: Social Game

- Basic version implemented within 1 week
- May quickly become popular
- Grows to millions of users in 2 weeks
- Lose its edge in 3 months and stagnate
- Users tolerate downtime and bugs
- Development pace is paramount
- Such systems often solve with simple design and fast implementation. Improving performance as needed.

# MySQL or NoSQL

- Optimal architectures often include MySQL and NoSQL components
- NoSQL solutions often scale well
  - Large volume/traffic key value storage
  - Analytics
  - Supplemental storage, buffering, caching
- MySQL is
  - Proven and reliable
  - Tools for monitoring, backups, recovery
  - SQL is well understood and expressive

# Silver bullet for MySQL ?

- General purpose SQL Clustering is hard
  - There are some solutions on a way
- MySQL Cluster
  - Hard. No MVCC
- Spider, Spock Proxy, Hscale
  - Not General purpose solutions, mostly “hacks”
- InfiniDB
  - Helpful for scaling Data Analytics
  - Early version (limited functionality)

# How good is Single Server

- Hardware is powerful these days !
- Compare to 2005 – start of sharding
- Commodity Servers can
  - Have 384GB of Memory vs 16GB
  - Have 32 CPU cores vs 4 cores
  - Per core performance is about 2 times faster
  - SSD vs Disk
    - Up to 100x+ more IOPS and 100x lower latency compared to single disk for FusionIO cards.
- Single Thread CPU performance posted lowest gains.

# You can get a lot from server!

- You can store hundreds of GB in memory
  - Working set for typical 1TB database size
- SSD can take care of high volume random writes
- SSDs make warmup pains significantly lower
- You can have:
  - 100K+ simple lookups/sec
  - 20K+ simple updates/sec
  - Over 5.000.000 rows traversed per second
- For many OLTP applications it is more than enough

# So our assumption is

- Your application is **LARGE SCALE**
  - Operating on large data set
  - Has very high transaction volume
  - Requires large data crunching with low latency
  - ... or may become such
- So you really need to scale beyond one server
- Less than 1% of applications are such
  - But these are cool applications so they get a buzz
  - They get larger proportion of developers and traffic

# Growth Choices with MySQL

- It often starts with Single Instance
  - Fast Joins, Ease of retrieval, Aggregation etc
- Becomes limited by CPU or Disk IO capacity
  - And do not forget about MySQL's internal scaling issues (problems with too many CPU cores, etc)
- “Scale-UP” is limited and expensive
  - Especially when it comes to “single thread” performance
- Simple next choices:
  - Caching, Buffering, Queuing
  - Functional Partition
  - Replication

# Performance Optimization Basics

- Best way to optimize something is stop doing it
  - Make sure you only get the data you need.
- If you can't avoid doing it cache it
- In any case consider batch it/buffer it
  - Round trips are expensive
- If possible delay/do it in the background
  - Queueing
- A lot can be done in addition to Optimizing operation!

# Performance and Complexity

- The highest performance/scalability often comes with increased complexity
  - But you can have slow AND complex too
- Ready solutions – built-in clustering, caching are very attractive.
  - Consider transparent web cache
- Understand the cost to perform and maintain optimization vs your performance needs
- Some applications choose not to do advanced caching, sharding, using slave etc.

# Caching Basics

- Cache hit on highest level is best
  - Browser cache
  - Squid Cache
  - Memcache
  - Database buffer cache
- Cache hit needs to be a lot cheaper than miss
  - 10ms Disk read at Squid Cache vs 5ms generation
- Cost of having cache vs using resources for other purposes

# Cache Basics

- Cache hit ratio needs to be high
- Cost of update/invalidate should be reasonable
  - Look for side effects. MySQL Query Cache limits scaling
- None or Simple application change for caching
  - MySQL Query Cache is fully transparent
- Caching should not affect user experience
  - At least users should not be annoyed by the change
- Cache High Availability
  - If you depend on cache you can't have it go down

# Caching Policies

- **Time Based**
  - This item expires in 10 minutes.
  - Easy to implement. Few objects cachable
- **Write Invalidate**
  - Changes to object invalidate dependent cache entries
- **Cache Update**
  - Changes to object cause update dependent cache entries
- **Version based Caching**
  - Check actual object version vs version in cache.

# Active vs Passive Cache

- **Active Cache**
  - Transparent. Will automatically generate data on cache miss.
  - MySQL Query Cache, some API driven development
  - Complex Cache. Easy to use.
- **Passive Cache**
  - Will return “miss” if data is not found
  - Need to manage cache with data updates
  - Simple Cache. Hard to use.

# Pre-Generation

- Form of Caching too
  - Assumes misses do not exist
- Can take form of Summary tables, memcache, files
- Periodic re-generation or updates on changes
- Easy to use from the application
- Very helpful if cache miss is not acceptable
  - Generating value takes too much time

# What To Cache

- Large Object (ie HTML page)
  - High Efficiency
  - Any change invalidates whole object
  - Many object variations to cache
- Small Object (ie Blog Comment on the page)
  - Work (CPU time) needed to create the Large Object
  - More complicated code
  - Need many “gets” to cache Hint: Multi-Get
  - More local invalidations
  - Less memory needed for caching.
- Try caching pre-processed data as possible

# Where to Cache

- **Browser Cache**
  - TTL Based
- **Squid, Varnish**
  - TTL, eTag, Simple invalidation
- **Memcache**
  - TTL, Invalidation, Update, Checking version
- **APC/Local in process cache**
  - TTL, version based, some invalidation

# Operational Challenges

- High Availability/Fault Tolerance
- Resizing Caching Tier
- Incompatible code changes
- Dealing with stale cache/selective invalidation
- Warmup
- Race Conditions
- Cache storm with miss on common object
  - Multiple request may be executed at the same time.

# Batching

- Less round trips is always good
- Think “Set at once” vs Object at once
  - `get_messages()` vs `get_message`
- Set API allows to parallelize operations
  - `curl_multi` to fetch multiple URLs in parallel
- Goes hand in hand with buffering/queuing
- There is optimal batch size
  - After which diminishing returns or performance loss

# Buffering

- May be similar to batching
  - Accumulate a lot of changes do them at once
- Also could aggregate changes, doing one instead of many
  - Counters are good example
- Buffering can be done
  - Inside process; File; Memcache; MEMORY/MyISAM table; Redis etc
  - Pick best depending on type of data and durability requirement

# Queuing

- Doing work in background
  - Helpful for complex operations
- Using together with Buffering for better performance
- Message Delivery
  - Cross data center network is slow and unreliable
- Generally convenient programming concept
  - Such as Job Management

# Software for Queueing

- A lot to chose from depending on needs !
- Simple Files
- MySQL Table
- Q4M (MySQL Storage Engine)
- Redis
- Gearman
- RabbitMQ
- ActiveMQ

# Background Work

- Two types of work for User Interaction
  - Required to Generate response (synchronous)
  - Work which does NOT need to happen at once
- User Experience design Question
  - Credit card charge confirmed vs Queued to be processed
  - Report instantly shown vs Generated in background
  - Youtube Videos are processed in background
- Background activities are best for performance
  - A lot more flexible load management
  - Need to ensure behavior does not annoy user.

# Queuing and Buffering

- Multiple tasks in the queue
- Higher load = larger queue
- Larger queue = better optimization opportunities
- Intelligent queue processing
  - Picking all tasks related to one object and processing them at once
  - Process all reports for given user at once. Better hit rate

# Message Delivery

- Communication cross data centers/servers in one
- Communications in Periodically connected environments
- MySQL Replication – no conflict handling
  - Though can be used for message delivery
- Message handling may go beyond database changes.

# General Programming

- Job Processing
  - Gearman is a great tool
  - Some jobs may be synchronous too !
  - Includes Scheduling/Failure handling too
- A lot of simple queue applications
  - Moderation queue
  - Crawling
  - Building Reports
- Additional Queue Programming primitives
  - Waiting on element to appear in the queue
  - Waiting on free space in full queue

# Distributing Data

- Replication
  - Multiple data copies. Scaling Reads
- Functional Partitioning
  - Splitting data by function. Scale reads and writes
- “Sharding”
  - Horizontal partition. Different users on different nodes. Scale reads and writes.

# Functional Partitioning

- “Let me put forums database on different MySQL Server”
  - Picking set of tables which are mostly independent from the other MySQL instances
  - Light duty joins can be coded in application or by use of Federated Tables
- Challenges
  - These vertical partitions tend to grow too large
  - And further vertical partitioning becomes complicated or impossible.

# Fault Tolerance

- Functional Partitioning – larger chance for one of components unavailable
- Replication/DRBD/etc to keep component available
- Designing application not to fail if single component does not work
- No need for all web site to be down if forums are unavailable
  - Even if last forum messages featured on the front page
- Design application to restrict functionality rather than fail.

# MySQL Replication

- Many applications have mostly read load
  - Though most of those reads are often served from Memcache or other cache
- Using one or several slaves to assist with read load
- MySQL Replication is asynchronous
  - Special care needed to avoid reading stale data
- Does not help to scale writes
  - Slaves have lower write capacity than master because they execute queries in single thread, and writes are duplicated on every slave
- Slave caches is typically highly duplicated.

# Taking care of Async Replication

- Query based
  - Use Slave for reporting queries
- Session Based
  - User which did not modify data can read stale data
  - Store binlog position when modification was made
- Data Version/Time based
  - User was not modified today – read all his blog posts from the slave
- MySQL Proxy Based
  - Work is being done to automatically route queries to slave if they can use it

# Slave HA and Load Balancing

- **MMM/ Virtual IP Based**
  - Especially if you have just 1 slave
- **LVS/HAProxy/ Hardware load balancing**
  - Health check can be designed to check status of replication.
- **DNS Based**
  - Beware of DNS caching issues, pools

# Replication And Writes

- Very fast degradation
  - Master 50% busy with writes. 2 Slaves have 50% room for read queries
    - 1 “Server Equivalent” capacity for the slaves
  - Master load grows 50% and it becomes 75% busy. There is 25% room on each of the slaves
    - Both Slaves are now equivalent to  $\frac{1}{2}$  of “Server Equivalent”
- Single Thread Bottleneck
  - Use single CPU
  - Submit single IO request at the time (most of the time)
  - Getting more common as servers get more cores

# Optimizing MySQL Replication

- Use “Percona” Patches to identify which queries are limiting replication performance
- “Row Level” replication in MySQL 5.1
  - No need to search for rows to update on the slave
- Replace complex INSERT/UPDATE statements with select and update
  - **INSERT ... SELECT** <very complex query>
  - Changing to:
    - **SELECT**
      - <store resulting rows>
    - **INSERT ....** <stored data>

# Reducing Impact of Single Thread

- Understand if CPU or IO is limiting factor
- Fast CPU cores, instead of many of them
  - Slave thread will not use many anyway
- Good cache fit is very important
  - Helps both with IO and CPU Usage
  - **Mk-slave-prefetch** can help by prefetching
- Batch queries if possible
  - Single row updates are expensive to replicate
- Flash Storage can help for IO bound replication

# Cross Data Center Replication

- Can shift bottleneck
  - From applying Binary logs to copying them
- Seconds\_Behind\_Master is unreliable
  - In any case, but especially in WAN replication
  - **Mk-heartbeat** is great for real lag monitoring
- Helpful Tips
  - VPN can help with security, and compression
  - **MASTER\_SSL=1** - native MySQL SSL
  - **Slave-compressed-protocol=1**
    - Global setting only.

# Minimizing Replication Latency

- Single Thread – Long Queries block the flow
- Query Chopping
  - **DELETE ... LIMIT 100** in the loop.
  - Goes well with separating select and update
  - Note these has to be **different** transactions.
- **ALTER TABLE** - Do it locally
- Use Helper for Complex operations (be careful)
  - Master inserts the “task” in the queue table
  - Script looks at the table and executes task on each slave
    - You also can control which slaves do it and which do not
      - For example keeping archive on some slaves.

# Replication and Caching

- Imagine you have 20GB database on 16GB Box
  - It almost fully fits in memory and you're only doing reads.
- Your database grows to 100GB and you add 5 slaves
  - However now each slave fits less than 1/5 of the database in memory and load becomes IO bound.
- You can improve it but never get it perfect
- There is storage duplication too
  - Fast Disk storage is not so cheap
  - And if you're using SSD this is very serious issue.

# Improving Replication Caching

- Slave Roles
  - Slaves for reporting queries
  - Slaves for Full Text Search
- Query Routing
  - All queries for user session go to the same slave
  - Even user\_id go to one slave odd to other
- Hard to avoid overlap fully
- Writes themselves have same working set on all slaves

# Different Schema

- You can have Different Schema on Master and Slave
  - Use extreme care using this. You've been warned
- Different indexes on Master and Slaves
  - Query mix can be different
- Different Partitioning settings
- Different Storage Engines
- Extra columns on the slave
  - For example containing cache
- **This is high powered medicine, beware of possibly lethal side effects.**

# Slave Operation Issues

- Warmup issues if promoting Slave to Master
  - Select mirroring with **mk-query-digest** –execute
- Accidental writes to the slave is common issue
  - Use **–read-only**, restrict SUPER privilege
- How to easily clone slave from the master ?
  - LVM snapshot
  - Xtrabackup/InnoDB Hot Backup for InnoDB only.
- Backups from the slave
  - Make sure to ensure Slaves match master

# Ensuring Replication is in Sync

- Replication can run out of sync
  - Writing to the wrong slave from Application
  - Operational Errors
  - MySQL Replication bugs
  - Master/Slave Crashes
- Validate replication consistency regularly
  - **Mk-table-checksum** is a great online tool

# ROW or STATEMENT replication

- ROW level replication new in MySQL 5.1
  - Is relatively mature by now.
  - It is not as transparent as Statement Level
- ROW level is sometimes more reliable
  - Required for replication to work with some features
- Performance gain vary
- ROW level generally uses less CPU on the slave
- **slave\_exec\_mode=IDEMPOTENT**
  - Allows using “approximate” binlog positions relatively well.

# Sharding

- When functional partition and replication can't help
- Breaking data in smaller pieces and storing them on the different servers (“clusters”)
- The “only” solution for very large scale applications
- Needs careful planning
- Can be hard to implement
  - Especially if application is not designed w sharding in mind
- How to “shard” the data is crucial question
  - And there could be multiple copies of data split by different criteria.

# Sharding and Scale

- Often Sharding is used for application of small scale
  - Complicating things beyond the need
- Hardware is Improving
  - When LiveJournal did Sharding 4GB was commodity
  - Now 256G+ of “cheap” Memory
- Decision for Sharding
  - Single Box Performance
  - Replication Capacity
  - Maintenance/Operations
    - 5TB Innodb table is a problem even if it performs well enough

# Sharding and Replication

- Sharding typically goes together with replication
  - Mainly for achieving high availability
  - DRBD, SAN is rarely used option
- One server crashes once per year
  - 50 servers – one crashes each week
    - And making data unavailable for portion of the customers
- We like Master-Master replication for ease of use
- Replication solves operational issues
  - How to upgrade/replace hardware/OS ?
  - How do you ALTER/OPTIMIZE MySQL Tables ?

# Sharding and Number of Slaves

- Symmetrical Master-Master is good base option
  - Only one Master is written at the same time
  - Same data center or different data centers
- Second Master can be used for
  - Just Redundancy purposes.
  - Reporting Queries, Business Intelligence, Scripts.
  - Portion of Read traffic
- Additional Slaves
  - Additional Redundancy or extra Read capacity
  - Slave for Disaster Recovery (different data center)
  - Delayed replication Slave

# How to shard the data ?

- Most of queries can be run within same shard
- The shard size does not go out of control
  - Good: Sharding Blogs by `user_id`
  - Bad: Sharding by `country_id`
    - Large portion of traffic can be from the same country
- Multiple splits at the same time possible
  - By Book at the same time by User
- Store full data in secondary sharding or only pointer/partial data

# Sharding Techniques

- Fixed hash sharding
  - Even ID go on **Server A**, odd on **Server B**
  - Inflexible. Though can be made better w consistent caching.
- Data Dictionary (typically best)
  - User 25 has his data stored on **Server D**
  - Flexible but dictionary can become bottleneck
- Mixed Hashing
  - Objects hashed to large number of values which mapped to servers
- Direct Path reference - `<shardid><objectid>`

# Gradual Sharding

- Sharding accesses to some of the data
  - Time pressures do not allow to shard everything
- May be mixed with functional partitioning
- Replication Topology
  - Main Master->ShardMasters->ShardSlaves
- Best if using as transition structure only
  - Hard to maintain.

# Tables and Shards

- Each UserID goes to his own group of tables (or database)
  - Too many tables if many users.
- There is single set of tables per server
  - Tables can get large.
  - Harder to move tables around servers
  - Easier migration for old applications
- Somewhere in between
  - Many Users per table group; many table groups per server
  - Flexible but a bit harder to implement

# Multiple MySQL Instances ?

- **Benefits**
  - Multiple Replication threads
  - Each instance is smaller size
  - Help with MySQL scalability on multi core
- **Drawbacks**
  - Harder to maintain
  - Loss of correlation to OS level
  - Different instances impact each other
- **Virtualization is another alternative to multiple instances.**

# Capacity Planning

- Good if you can dynamically add/enable shards
- Leave Space for the growth
  - You often know how many “objects” per shard perform well
- Consider historical data use pattern
  - For example many users may be “playing” for month with system and when leaving
- Consider data growth and their access pattern
  - May be most accesses happen to the last month of data
- Moving objects between shards is likely to be needed.

# Data Archiving

- Sometimes in addition to sharding by object sharding by time is used
- Old data can be stored on archive servers
  - le messages over 3 months ago almost never accessed
- Full archiving or “keeping the headers”
- Often dictionary modification with “cutoff date” for use of archive server is used.
- Archiving can be done to non MySQL system all together.

# Moving data between Shards

- Sooner or later needed to balance the load
- Moving by one object
  - Temporary marking this object read-only
    - Can avoid but too complex so mostly impactful
  - Moving many objects takes a lot of time
  - Minimal system impact
- Moving by table/database
  - Easy (standard tools like mysqldump) and quickly
  - Larger system impact
    - As whole table groups need to be made read only.

# What Takes care of Sharding

- Database Access Layer
  - Easier if you start developing with shards in mind
- Database Access Layer query parsing
  - Extract **user\_id=X** from query and route it as needed.
- HiveDB <http://www.hivedb.org>
- HSCALE <http://www.hscale.org>
- Spock Proxy
- Some development in MySQL Proxy
- DMP
- We can see there is no common solution still

# Accessing Global Data

- You may need to “JOIN” data w some global tables
  - User information, regions, countries etc
- Just join things Manually
  - Also makes caching these items more efficient
- Replication of global tables
  - Could be MySQL replication or copy for constant tables.
- Access via Federated Storage Engine
  - Be careful, but works for light duty join
  - Adds challenges with HA provisioning

# Accessing Multiple Shards

- Global Search, Analytics, Rating, “Friends Updates
- Accessing few shards or Accessing All Shards
  - Think about these type of needs designing sharding
- Creating Summary Tables
- Parallel execution of queries on multiple shards
  - Can be tricky to do in some programming languages
- Loading data for analytics
  - Hadoop, Kickfire etc.
- Using other software
  - Sphinx, Lucene etc

# Caching & Replication

- Caching/Buffering – get more capacity out of a system
  - It is possible to get 10x+ load reduction
  - There are often low hanging fruits
  - Advanced Caching causes code complexity
    - Similar to functional partitioning
- Sharding allows scaling to 100s of servers
- Scale requirements and cost guide what matters most
- Is not black and white
  - Simple Caching->Sharding->More Caching

# Consider Combined Effect !

- Consider combined effect when growth planning:
  - Hardware upgrade 3x
  - Functional Partitioning 2x
  - Replication 3x
  - Caching 3x
- Total Gain: 50x
  - Without need of expensive sharding !

# Thanks for Coming

- Questions ? Followup ?
  - [pz@percona.com](mailto:pz@percona.com)
- Yes, we do **MySQL and Web Scaling Consulting**
  - <http://www.percona.com>
- Check out our book
  - Complete rewrite of 1<sup>st</sup> edition

