



PERCONA
Performance Consulting Experts

Scaling Applications with Caching Sharding and Replication

April 11-14, 2011

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

Who Are You ?

- MySQL Developers ?
- MySQL DBAs ?
- System Administrators ?
- System Architects ?
- Managers ?

Do you already use

- MySQL ?
- Multiple MySQL Instances ?
- Replication ?
- Sharding ?
- Memcache ?
- NoSQL Solutions ?
-

How large is your application ?

- How many user interactions do you have per second at peak time ?
 - Less than 10 ?
 - 10-100 ?
 - 100-1000 ?
 - 1000-10000 ?
 - 10000+ ?

How Large is your Total Database

- Total Data size you have in your system, excluding replicas ?
 - Less than 10GB
 - 10GB to 100GB
 - 100GB to 1TB
 - 1TB to 10TB
 - 10TB+

How large do you want to be ?

- Everyone wants to be like Google or Facebook
 - But few people really need to be
- Many projects dream about large scale but will take years and a lot of luck to reach it
- Large Scale Architectures come with complexity
 - Development, operation etc
 - They down the development process in most cases

The Skew

- Less than 0.1% of MySQL Applications are “large scale”
- Companies owning these applications employ larger proportion of MySQL DBA's and Developers (especially good ones)
- People Blog Disproportionally about cool stuff
 - Which is often large scale and complicated
- A lot of large scale companies based in SF Bay Area/Silicon Valey

Architecture Questions

- How much growth in load and database size do I expect
 - Looks for higher estimates
 - “What if Application would just stop functioning if these numbers are exceeded” ?
- How Likely are those numbers and when we'll know better ?
- The Life time of the Architecture
- The Complexity of Changes
- Risk Tolerance

Simple Realities

- 37 signals avoiding sharding, replication, caching
 - Replication is used but for failover only
 - Hundreds of thousands of users
 - Better pay for more powerful hardware but launch new features faster
- Some well known Enterprise Company
 - 200K employees
 - Using Drupal based application daily
 - Single MySQL instance is less than 25% loaded
 - Built in memcache caching is used in this case.

Modern Hardware Realities

- The “Affordable” Server these days can be
 - 48+ Cores
 - 256+ GB Of Memory
 - Flash storage providing 100K+ IOPS
 - You can reach over 1M IOPS per server these days
- You can serve 100K+ queries/sec on such system
 - And store 1TB+ Data

Lets Do Some math

- Store Working set for 1TB database in memory
 - In somewhat typical application
- We can get
 - Over 200K+ simple lookups per second
 - 20K+ simple update per second
 - 5.000.0000+ rows traversed per second
- Assuming we have 100 read queries + 10 write queries, 2500 rows read per user interaction
 - 2.000 user interaction/sec from single server.
 - 80M interactions/day (allowing for peak)

Choose Hardware Right

- Avoid Being too cheap or taking someone else's recipe
- Example 1:
 - Using 16G of memory. 64G allows to server 10x traffic
- Example 2:
 - Using shelf of 2.5K 15K RPM SAS. Flash is a lot faster.
- Example 3:
 - Buying system with many slow cores
 - MySQL Replication would not scale.

True Cost of Hardware

- Does 50K server looks expensive ?
 - It well may be !
- But put this in prospective
 - Cost of complicated development ?
 - Cost of longer time to maket ?
 - Cost of software & operational mistakes bugs ?
 - Cost of more expensive operations ?
 - Space and power cost for many cheap servers

Optimize when Scale !

- Most people go to complicated architecture without getting all performance from current system
 - Tuning Settings
 - Query Optimization
 - Schema Optimization
- Which of them gives the most improvement ?

Do hard changes if you have to

- In many cases it is “too hard” to do certain changes
- So people migrate away from MySQL to something else
- As Part of “migration” architecture is changed completely
- This is “sold” as MySQL Being “unable to handle the load”
 - This may be the only way to get a resources needed for redesign in your organization but do not trick yourself

Impact of Growth

- Number of Queries
- Database Size
- Query Execution Complexity
- Changes in system use
- These will vary depending on application
 - Make sure to consider it right while planning

Item at Once vs Set at Once

- Architecture Designed One item at the Time
 - Inefficient
 - Often Sequential Processing, hard to do in parallel
- Architecture Designed to process Set at the time
 - Can be a lot more efficient
 - Consider multi value insert
 - Parallel processing can be added easier
- Examples
 - `get_multi` in memcache
 - `curl_multi`
 - `SELECT IN (...)`

How to Optimize Something

- The best way to optimize something is stop doing it
 - Trivial but most of applications execute queries and not use all results.
- Caching and Buffering
- Reschedule
 - Run at the time when there are more resources
 - We do not run backups in peak time, right ?
 - Run at the different place
 - Slave ?
- Optimize/Eliminate Waste

Caching

- Getting result from “somewhere” instead of generating
 - Many layers of caching
- Pre-Generation
 - Caching with no misses
- Caching Problems
 - Cost of maintaining cache
 - Dealing with Stale data in Cache
 - Impact of Cache Misses

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.

What can be used for Caching ?

- **Static Files**
 - Great if you can serve them directly to the clients
- **Memcache**
- **Database Tables**
- **In Process cache**
 - Directly in Java App
 - APC/Xcache etc for PHP
- **Squid/Varnish**
- **Set damn HTTP expire headers !**

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
- In memory buffer also works

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
- Load Management
 - Deal with short spikes without overloading system
 - Prioritize what is more important

Consider Direct NonSQL Access

- It is SQL Part which is the problem
 - Handling many connections
 - Overhead of Parsing, locking tables etc
- MySQL Cluster (NDB)
 - You can use native API which is a lot faster
- Using Innodb
 - HandlerSocket (included in Percona Server)
 - Get up to 700K read transactions/sec per node.

So What is your problem ?

- Response Time
 - Query takes longer than needed to respond
- Capacity
 - Can't run as much QPS as I'd like
 - Or can but response time suffers

Solving the Problem

- Capacity
 - You can find what queries put the most load on the system
 - Check out mk-query-digest and deal with them
- Response Time
 - Optimize The query
 - Make it parallel
 - Pre-Create result
 - You may need to redesign so you do not need such a query

Eliminating Waste

- Are all results fetched by client used
 - All columns in all rows ?
- Did MySQL only analyze rows needed to create result set ?
- Were only needed reads from disk performed?
 - Or are you caching stuff you really need to cache
- Is the data read is data you use
 - Rows/columns which are not being used
 - Empty/deleted space

What if my query scans 10M rows

- Query with intrinsic complexity
- Do you really need it, ie exact count(*) for result ?
- Is it possible to pre-create or cache ?
 - Watch for response time if caching expensive queries
- Can you run it in parallel on same or many nodes
 - Jobs with Gearman
 - “Shard Query” by Justin Swanhart
 - <http://www.mysqlperformanceblog.com/2010/11/15/shard-query-adds>
 - MySQLnd running queries in parallel
 - Java Threads, multiple processes, curl_multi etc

Going Beyond single server

- Replication
 - Scale reads
 - Restrict Writes
- Functional Partitioning
- Sharding

Replication

- Allows to Scale Reads well
- Asynchronous
 - Reading potentially Stale data
 - Semi-Synchronous option in MySQL 5.5
 - Still can read potentially stale data
- Data Duplication
 - High cost for storage (especially SSD)
 - Duplication of Cache too
 - Can reduce but impossible to eliminate
- Single Thread
 - See next slide

Single Thread Replication

- Limited by Replication sooner than by Master Write Capacity
 - More and more problem as we have many CPU cores and hard drives
 - Good to use master for some reads too to avoid waste of resources
- Parallel replication
 - Tungsten by Continuent has some support (Open Source)
 - Drizzle looking to have one

Replication and Caching

- Many applications are mainly reads
 - Using Slaves to assist with reads is a great idea
- Caching however can take care of reads too
 - Reducing need for slaves
- Better your caching, less slaves you will need

Replication Performance

- Know it !
 - 10% load increase can push you from “never lags” to “never catches up
 - May happen with database grow and “same load as yesterday
- Percona Server has instrumentation to show replication thread load
- Pause Replication; Measure how long it takes to catch up
 - Strive for 1:2 or better 1:3 ratio for catchup time vs paused time

Improving Replication Performance

- Slave often is very slow to warmup caches
 - Single thread hurts here as well
- Consider Fast Warmup available in Percona Server
- Find what queries are loading replication and fix them
 - **--log-slow-slave-statements**
 - Mk-query-digest to analyze the log
- Mk-slave-prefetch
 - Can help to get a bit more replication performance

ROW or STATEMENT replication

- STATEMENT based replication is “Traditional” for MySQL
 - Very compact for statements changing many rows
 - Has to perform complete query execution on the slave
 - Problem if finding rows more costly than updating
 - Requires more locking
 - Does not work for all queries
- ROW Based (MySQL 5.1+)
 - See next slide
- Check for number of other differences
 - Such as trigger handling

ROW Based Replication

- Can generate larger binary log files
- Confusing for troubleshooting
 - Error messages are a lot more cryptic
 - “Could not execute Update_rows event on table repl.t1; Can't find record in 't1', Error_code: 1032..”
 - Slow query logging not helpful
- Can be more efficient
 - Queries spending a lot of time finding rows
 - CPU bound workloads
- More Efficient locking, Works with all statements, Has IDEMPOTENT mode.
- More restricted in Different Schema support

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.

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.

Replication Cache Duplication

- 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 Cache Duplication

- **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

What else replication is good for

- In many cases you want replication even if you do not need it to scale
- High Availability
- DR (Slave in the different data center)
- Analytics
- Development
- Backups
 - Make sure replication is in sync
- “Online” Upgrade and schema maintenance
- Check MMM to manage master-master pairs

Functional Partitioning

- Place different “functions” on different database servers
 - Hint – design system without adding data dependencies without a good need
- Main Web Site/Forum/Blog may all use independent database
 - The light dependencies can often be coded around
- Problem: There are not so many functions around
 - Also one function is likely to be responsible for large portion of load

Fault Tolerance

- How your application behaves if one of components fail ?
- Is in addition to **high availability** which reduces probability of such failure.
- More components increases probability at least one of them is down
- Application should continue to function if some useful functionality remains
 - Google Search can stop adding News, Videos to main search if they have problems.

Sharding

- Sharding can be easy and can be hard
- SaaS companies – independent customers on independent boxes
 - Some do not even call it sharding
- Easy Sharding
 - Does not add much development and operation complexity
 - Do it early in life
 - Different databases on same server is good enough
 - Allows to avoid adding dependencies without a thought

Hard Sharding

- For most apps Sharding is not easy
 - Adding sharding results in significant complexity increase
- Think about sharding
 - Avoid decisions you'll later regret
- Do not rush into the sharding unless you clearly need it
 - Important to do capacity analysis/planning to know how much you've got left with current architecture

Reasons to Shard

- Single box (Instance) Performance
- Replication Capacity
- Maintenance/Operations
 - Dealing with 5TB Innodb problem can be painful

So you've decided to shard

- Preserve locality access when possible
 - Shard by “user” - many accesses are local
- Even larger shard should be manageable
 - Sharding by “country” is a bad idea
 - State ? City ? May be good for some local sites
- May need to double store the data
 - Shard by X and Y
- Queues are good for cross shard synchronization
 - Things should not break if one shard is down

Type of Sharding

- **Hardcoded**
 - “Even on one server odd on another”
 - Avoid in most cases. Painfully inflexible
- **Dictionary**
 - Some main database serves as location directory
 - Can be also used to store flags like “unavailable”, “RO”
 - Is highly cachable – easy to scale
- **Exotic**
 - There are other ways, though they are rather rare

Where to Implement Sharding

- Application database access layer
 - Some work but but good transparency
- ORM framework may support it
 - Good if it does it well.
- “Proxy”
 - Spock Proxy; Shard Query etc
 - Nice and transparent. Understand what you're doing
 - Commercial Solutions such as ScaleBase

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

Sharding and Tables

- “Database Per Sharded Key”
 - Good if there are few “users” (Enterprise SaaS)
- One database per Instance
 - Especially for sharding retro-fits
 - Less changes for the code
 - Database can grow pretty large
- Multiple tables, each having many keys
 - “Many Shards Per Box”
 - Flexibility.
 - Good choice for sharded design

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.
- Replication based Shard Splitting

Replication Based shard Splitting

- Make Slave 1A and 1B for Shard 1
- Set them up with usual number of slaves
- Re-Configure mapping so some traffic goes to Shard 1A, other 1B
 - Requires very brief downtime
- Gradually Purge data from 1A and 1B which does not belong there

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 !

Alternative Solutions

- MySQL Cluster
 - Getting better for general purpose use
- Schooner MySQL
 - Synchronous Replication
- Clustrix
 - Cluster aware optimizer/engine
 - Can be talked to using MySQL Protocol
- ScaleDB, GeneDB etc

Percona Live, May 26, New York



www.percona.com/live

Thanks for Coming

- Questions ? Followup ?
 - pz@percona.com
- Yes, we do **MySQL Support, Consulting, Training**
 - <http://www.percona.com>
- Check out our book
 - Complete rewrite of 1st edition

