PERCONA

Performance Consulting Experts

Scaling MySQL-powered Web Sites by Sharding and Replication

June 24, 2008

Velocity Conference

Burlingame,CA

by Peter Zaitsev, Percona Inc

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 accelerated 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 lie 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 to expensive and you may never get the product to the market.

But is not there a silver bullet ?

- MySQL Cluster ?
- Continuent/Sequoia ?
- KickFire ?
- MySQL Proxy ?
- BigTable ?
- SimpleDB ?
- All have their limitations in scaling or ease of use
  - And you better know these in advance

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:
  - Vertical Partition
  - Replication

Vertical 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.

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
- 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

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 growths 50% and it becomes 75% busy. There is 25% room on each of the slaves
    - Slaves are now equivalent to ½ of “Server Equivalent”
- Single Thread Bottleneck
  - Use single CPU
  - Submit single IO request at the time (most of the time)
  - The problem on medium/high end servers mostly

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 growths 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

Sharding

- When vertical partition and replication can't help
- Breaking data in smaller pieces and storing them on the different servers
- The “only” solution for 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 Replication

- Sharding typically goes together with replication
  - Mainly for achieving high availability
- 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 ?

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
  - 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>

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

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

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
  - Do you have spare Netezza or Kickfire around ?
- Using other software
  - Nutch, Sphinx, Lucene etc

Caching

- How do not I say anything about caching ?
- Caching is must have for large scale web app
- May reduce your database performance demands 10x+
- Only delay the time when you need to get things sharded and replicated

Thanks for Coming

- Questions ? Followup ?
  - pz@percona.com
- Yes, we do **MySQL and Web Scaling Consulting**
  - http://www.percona.com
- Check out our book
  - Just came out last week
  - Complete rewrite of 1st edition