Performance Best Practices For MySQL

Apr 11-14, 2011
MySQL Conference and Expo
Santa Clara, CA
by Peter Zaitsev, Percona Inc
The Audience

- How many of you are?
  - DBAs?
  - Sysadmins?
  - Developers?
  - Architects?
  - Managers?
About Presentation

• An overview of Performance Related Practices
• High level
• I will not teach you how indexes work or how to set innodb buffer pool size
  – Many presentations on the conference
  – “High Performance MySQL Book”
  – Percona has great training program teaching these practices
What Does it mean to have a good performance?

- You users know feel the answer
- Product manager may be asked the answer
- Developers are often left without an answer
- How many of your are ready to define what is considered good performance for your application?
It is about response time!

- Users feel response time, not throughput
- It is their perception of response time what is important
  - Database may or may not be significant component of it
- Stand alone outliers are not big deal
  - As network itself is not 100% reliable
  - Look at 99% or 99.9% response time
- Good performance for every user/use case
- Good performance all the time
- Response time requirement varies by action
Measuring Performance

- On the Client
  - Includes network time, page render, java script
  - Many tools. Jiffy-Web open source one

- On Your Web Server
  - Includes server side response time only

- Apache users can add Response time information to the log “%D”
  - And have it for each requests

- Logs are easy to load into database, Hadoop etc for processing
Instrumentation

• Where response time comes from?
  – CPU Time
  – “Waits” (such as MySQL, Memcache etc)

• Support information
  – User
  – Request_id (link multiple layers together)
  – User Interaction
  – Cache hit/miss info

• Check out PHP Instrumentation for example
  – http://code.google.com/p/instrumentation-for-php/
Response time and Throughput

• Response time
  – Always important for any application
• Throughput is not inverse of response time
  – Parallel execution
• The more load you put on the system the higher response time becomes
  – Peak throughput often reached with too bad response time
• Capacity
  – Throughput, at which response time is still acceptable
• Important for most multi user systems
Response time or Capacity

• Often Capacity is the problem
  – We can serve 1 concurrent user OK, 1 million is a problem
• Sometimes response time problem
  – This report takes 10 hours to generate for 1 user
• Solutions often different
  – Adding more CPU cores will not help to make single query to run faster in MySQL
Resource Hogs

• System works well for 999.999 users out of 1000000
  – Is it good enough?
• Not if that user is responsible for half of revenue
• And not if this user slows everyone down
  – Triggering wrong execution plan may make query to run
    1000000 times less efficient
• You need to optimize system to deal with such outliers
  – Or protect the system to get rid of them
  – This is where cooperation with engineering and business is important
General Architecture Goals

- Cost (Efficiency)
- Performance
- Scalability
- High Availability
- Development Ease/Cost
- Operational Ease/Cost
Efficiency, Scalability, Performance

• Does Scalable mean Efficient?
• Yes it scales to 10000 nodes but it takes 20 nodes to reach performance of 1 node?
  – You need to know how much scalability do you need
• Your performance goals and scale will define what is the most efficient solution and architecture for you
Replication, Caching, Sharding

• As long as they are not transparent they complicate development and operations
• Transparent can complicate operations too
  – More moving parts
• Replication
  – Dealing and managing slave lag
• Caching
  – Cache invalidation, priming cache etc
• Sharding
  – Synchronizing data, retrieving from multiple shards
Applications are different

- Replication, Caching Sharding can be easy or hard for your application
- Application scale defines how much “pain” do you need to go through
- Company business and culture is also important
  - Some prefer to build complex efficient systems
  - Others throw hardware on the problem
Prepare for Change

- Change is the only thing which is constant
- A lot of pain come from assumptions something does not change
- Especially when “big bets” are made in architecture
  - Sharing choices can be one such bet
- Product changes, use cases, data sizes, hardware etc
High Availability and Fault Tolerance

• High Availability – system just continues to work when component fails
  – RAID is a great example

• Fault Tolerance
  – What do we do if that highly available system failed?
  – Your bike may be your FT solution for your car
  – Design system component failures have as limited impact as possible.

• Large problem for 1% user or small for 100%?
  – You can provide a lot more care on business level to small portion of users
Growth and Performance

- Request Volume growth
- User Base Growth
- Data Size Growth
- Query Complexity Growth
- You need to understand how these are related for your system
  - It can be very different!
Powerful Hardware

• You may not realize how powerful hardware is these days
  – Especially if you're running in the Cloud, hosting
• Commodity Hardware Gets you
  – 48+ CPU Cores
  – 256+ GB of Memory
  – Flash Storage with 100K-1M IOPS
• Tens of thousands MySQL Queries per second
  – Sometimes more than 100K/sec
What can you fit on single box?

- 37 Signals runs number of applications from single box (millions of users)
  - They really dislike sharding
- The company with 200K Employees run Drupal powered Web Site based on MySQL
  - Actively used in business operation, set as start page in browser
  - Uses about 20% of Single Server Resources
- Single box is serving traffic in both cases,
  - There is Slave(s) for HA in both cases.
Commercial Solutions

• Numbers of companies are working to make MySQL Better
  – May be more cost efficient than complicated development for some customers
• Schooner Technologies
• Clustrix
• Infobright, InfiniDB – Column Store engines for Analytics
  – Both have restricted Open Source Versions too
Too Big To Fail

- Would you like to have all your data in very reliable 1PB cluster
- What if someone does wrong update?
- Large Scale Recoveries are hard
  - So providing care to all your users at once
- Consider using isolated systems when possible
- Do staged upgrades (software and hardware)
  - Sometimes staged code push make sense
Ways to Improve Performance

- Avoid Doing unnecessary stuff
- Caching and buffering
- Re-schedule (Time and Place)
  - Run complicated stuff at night or from the slave
- Optimize The query
Data Processing Paradigm

- Object at once vs Set at Once?
- Sequential or Parallel?
- Operating object by object sequentially is easy
- Working with sets is harder
  - But can be optimized and done in parallel a lot better
- Examples
  - Use multi value inserts
  - Memcache get_multi
  - Curl multi
Two types of complexity

- Lookup by primary key causing full table scan
  - Complexity as a waste. We only need to retrieve 1 item
- Complex group by on the same table
  - We need to process all data to provide result set
  - Intrinsic complexity
- Can just add index and eliminate complexity in 1st case
- Can only cache/builds summary/do in parallel/change algorithm in second
Capacity Planning

- How much load can current system handle?
- How large system do I need to build to handle future load?
- Can be very tricky in real world
  - Unknown user growth speed
  - Optimization efforts under way
  - Business is brainstorming on new product features
Benchmarking Challenge

• Build system for planned data size
• Drive estimated load on your system
• It is very hard to generate realistic data
  – Especially for features not launched yet
• It is hard to guess exact user use case
• Some systems make it easier than others
• Benchmarking is a very good thing to do
  – Just take it with grain of salt
What are you testing?

- Full Infrastructure?
  - Few companies can afford complete infrastructure for testing
    - And smaller scale versions may hide problems
  - Gives most reliable test results
- Single Database Box
  - Helpful if database is the only concern
  - A lot easier to do
- Load testing in production environment
  - During low time, carefully avoiding system to tip over
  - Can't benchmark any big changes this way
Trial by Sharding

- Can upgrade/adjust settings/ try different hardware
  - On single shard (or slave)
- Sometimes can select amount of load goes to shard or slave
- Very good data for existing application
  - How much performance is affected by data size and load
Testing with reduced resources

- Will my database handle double data size?
- I can test how database behaves with \( \frac{1}{2} \) of memory and CPU resources
  - Need to provide \( \frac{1}{2} \) of IO capacity or factor in disk utilization
- Can be very helpful when generating larger data and realistic traffic can be an issue
- Can be used “mirroring” traffic to slave
Resource Usage Analyses

- Query needs CPU, IO, Memory to Complete
- Plus there are internal limits like row level locks, latching etc
- If you know how many resources given query takes you can estimate how many queries server can handle
  - [http://www.mysqlperformanceblog.com/2011/03/08/](http://www.mysqlperformanceblog.com/2011/03/08/)
- Very helpful for systems with current low load
- Allows to do modeling query by query, easily play “what if”
The End

• Thank you!
• Send your feedback to pz@percona.com
• Percona Does MySQL Support, Consulting, Training
• We're creators of