Web Performance and Scalability with MySQL

Ask Bjørn Hansen
Develooper LLC
Real World
Web Scalability
MySQL Edition

Ask Bjørn Hansen
Develooper LLC
Hello.

- I’m Ask Bjørn Hansen
- Tutorial in a box 44 minutes!
- 53* brilliant° tips to make your website keep working past $X$ requests/transactions per $T$ time
  - Requiring minimal extra work! (or money)
  - Concepts applicable to ~all languages and platforms!

* Estimate, your mileage may vary
° Well, a lot of them are pretty good
Construction Ahead!

- Conflicting advice ahead
- Not everything is applicable to every situation
- Ways to “think scalable” rather than end-all-be-all solutions
Questions ...

• How many saw my talk last year?
• ... Brian Akers replication talk earlier today?
• ... Second Life talk a few hours ago?
• How many are using Perl? PHP? Python? Java? Ruby?
• ... Oracle? PostgreSQL?
• Lesson number 1

• Think Horizontal!

• Everything in your architecture, not just the front end web servers

• Micro optimizations and other implementation details — Bzzzzzt! Boring!
Benchmarking techniques

- Scalability isn't the same as processing time
- Not “how fast” but “how many”
- Test “force”, not speed. Think amps, not voltage
- Test scalability, not just performance
- Use a realistic load
- Test with “slow clients”
Vertical scaling

• “Get a bigger server”
• “Use faster CPUs”
• Can only help so much (with bad scale/$ value)
• A server twice as fast is more than twice as expensive
• Super computers are horizontally scaled!
Horizontal scaling

- “Just add another box” (or another thousand or ...)
- Good to great ...
  - **Implementation**, scale your system *a few* times
  - **Architecture**, scale dozens or **hundreds** of times
- Get the big picture right first, do micro optimizations later
Scalable Application Servers

Don’t paint yourself into a corner from the start
Run Many of Them

- For your application...
- Avoid having *The Server* for anything
- Everything should (be able to) run on any number of boxes
Stateless vs Stateful

- “Shared Nothing”
- Don’t keep state within the application server (or at least be Really Careful)
- Do you use PHP or mod_perl (or something else that’s running in Apache HTTPD)?
- You get that for free! (usually)
Caching

How to not do all that work again and again and again...
Generate **Static Pages**

- Ultimate Performance: Make all pages static
- Generate them from templates nightly or when updated
- Doesn’t work well if you have millions of pages or page variations
Cache **full** pages
(or responses if it’s an API)

- Cache full output **in the application**
- Include cookies etc. in the “cache key”
- Fine tuned application level control
- The most flexible
  - “use cache when this, not when that”
- Use regular expressions to insert customized content into the cached page
Cache full pages 2

- Front end cache (mod_cache, squid, ...) stores generated content
- Set Expires header to control cache times
- **or** Rewrite rule to generate page if the cached file doesn’t exist (this is what Rails does)
  - RewriteCond %{REQUEST_FILENAME} !-s
  - RewriteCond %{REQUEST_FILENAME}/index.html !-s
  - RewriteRule (^/.*) /dynamic_handler/$1 [PT]
- Still doesn’t work for dynamic content per user (”6 items in your cart”)
- Great for caching “dynamic” images!
Cache partial pages

- Pre-generate static page “snippets”
  (this is what my.yahoo.com does or used to do...)
- Have the handler just assemble pieces ready to go
- Cache little page snippets (say the sidebar)
- Be careful, easy to spend more time managing the cache snippets than you save!
- “Regexp” dynamic content into an otherwise cached page
Cache **data**

- Cache data that’s slow to query, fetch or calculate
- Generate page from the cached data
- Use the same data to generate API responses!
- Moves load to cache servers
  - (For better or worse)
- Good for slow data used across many pages
  (“today’s bestsellers in $category’’)
Cache hit-ratios

• Start with things you hit all the time
• Look at database logs
• Don’t cache if you’ll spend more energy writing to the cache than you save
• Do cache if it’ll help you when that one single page gets a million hits in a few hours
Caching Tools

*Where to put the cache data ...*
A couple of bad ideas

Don’t do this!

- Process memory ($\text{cache}\{\text{foo}\}$)
  - Not shared!
- Shared memory? Local file system?
  - Limited to one machine (likewise for a file system cache)
  - Some implementations are really fast
- MySQL query cache
  - Flushed on each update
  - Nice if it helps; don’t depend on it
MySQL cache table

- Write into one or more cache tables
- id is the “cache key”
- type is the “namespace”
- metadata for things like headers for cached http responses
- purge_key to make it easier to delete data from the cache

```sql
CREATE TABLE `cache` ( 
  `id` varchar(128) NOT NULL, 
  `type` varchar(128) NOT NULL default '', 
  `created` timestamp NOT NULL, 
  `purge_key` varchar(64) default NULL, 
  `data` mediumblob NOT NULL, 
  `metadata` mediumblob, 
  `serialized` tinyint(1) NOT NULL default '0', 
  `expires` datetime NOT NULL, 
  PRIMARY KEY (`id`,`type`), 
  KEY `expire_idx` (`expire`), 
  KEY `purge_idx` (`purge_key`) 
) ENGINE=InnoDB
```
MySQL Cache Fails

- Scaling and availability issues
- How do you load balance?
- How do you deal with a cache box going away?
- Partition the cache to spread the write load
- Use Spread to write to the cache and distribute configuration
MySQL Cache Scales

- Most of the usual “scale the database” tricks apply
- Partitioning
- Master-Master replication for availability
- .... more on those things in a moment
- memcached scheme for partitioning and fail-over
memcached

- LiveJournal’s distributed caching system
  (also used at slashdot, wikipedia, etc etc)
- memory based
- Linux 2.6 (epoll) or FreeBSD (kqueue)
  - Low overhead for many many connections
- Run it on boxes with free memory
- No “master”
- Simple lightweight protocol
  - perl, java, php, python, ruby, ...
- Performance (roughly) similar to a MySQL cache
- Scaling and high-availability is “built-in”
Database scaling

How to avoid buying that gazillion dollar Sun box

~$3,500,000
Vertical

~$2,000
( = 1750 for $3.5M!)
Horizontal
Be Simple

- Use MySQL
  - It’s fast and it’s easy to manage and tune
  - Easy to setup development environments

- PostgreSQL is fast too :-}
Replication

More data more places!
Share the love load
Basic Replication

- Write to one master
- Read from many slaves
- Great for read intensive applications

Lots more details in “High Performance MySQL”
Relay slave replication

- Running out of bandwidth on the master?
- Replicating to multiple data centers?
- A “replication slave” can be master to other slaves
- Almost any possible replication scenario can be setup (circular, star replication, ...)

![Diagram of relay slave replication](image-url)
Replication Scaling – Reads

- Reading scales well with replication
- Great for (mostly) read-only applications

(Thanks to Brad Fitzpatrick!)
Replication Scaling – Writes
(aka when replication sucks)

- Writing doesn’t scale with replication
- All servers needs to do the same writes
Partition the data

Divide and Conquer!

or

Web 2.0 Buzzword Compliant!

Now free with purchase of milk!!
Partition your data

- 99% read application? Skip this step...
- Solution to the too many writes problem: Don’t have all data on all servers
- Use a separate cluster for different data sets
- Split your data up in different clusters (don’t do it like it’s done in the illustration)
Cluster data with a master server

- Can’t divide data up in “dogs” and “cats”?
- Flexible partitioning!
- The “global” server keeps track of which cluster has the data for user “623”
- Only auto_increment columns in the “global master”
- Aggressively cache the “global master” data

![Diagram of cluster data with a master server]

- `select * from some_data where user_id = 623`
How this helps “Web 2.0”

• Don’t have replication slaves!
• Use a **master-master** setup in each “cluster”
• master-master for redundancy
• No latency from commit to data being available
• Get IDs from the global master
• If you are careful you can write to both!
  • Make each user always use the same master (as long as it’s running)
Hacks!

Don’t be afraid of the data-duplication monster
Summary tables!

• Find queries that do things with COUNT(\*) and GROUP BY and create tables with the results!

• Data loading process updates both tables

• or hourly/daily/... updates

• Variation: Duplicate data in a different “partition”

• Data affecting both a “user” and a “group” goes in both the “user” and the “group” partition (Flickr does this)
Summary databases!

- Don’t just create summary tables
- Use summary databases!
- Copy the data into special databases optimized for special queries
  - full text searches
  - index with both cats and dogs
  - anything spanning all clusters
- Different databases for different latency requirements (RSS feeds from replicated slave DB)
“Manual” replication

- Save data to multiple “partitions”
- Application writes two places or
- last_updated and deleted columns or
- Use triggers to add to “replication_queue” table
- Background program to copy data based on the queue table or the last_updated column
- Build summery tables or databases in this process
- Build star/spoke replication system
a brief diversion ...

**Running Oracle now?**

- Move read operations to MySQL!
- Replicate from Oracle to a MySQL cluster with “manual replication”
- Use triggers to keep track of changed rows in Oracle
- Copy them to the MySQL master server with a replication program
- Good way to “sneak” MySQL in ...
Make everything repeatable

• Script failed in the middle of the nightly processing job? (they will sooner or later, no matter what)

• How do you restart it?

• Build your “summary” and “load” scripts so they always can be run again! (and again and again)

• One “authoritative” copy of a data piece – summaries and copies are (re)created from there
More MySQL

Faster, faster, faster ....
Table Choice

• Short version:
  Use InnoDB, it’s harder to make them fall over

• Long version:
  Use InnoDB except for
  • Big read-only tables (smaller, less IO)
  • High volume streaming tables (think logging)
    • Locked tables / INSERT DELAYED
  • Specialized engines for special needs
  • More engines in the future
  • For now: InnoDB
Multiple MySQL instances

- Run different MySQL instances for different workloads
  - Even when they share the same server anyway!
- Moving to separate hardware easier
- Optimizing MySQL for the particular workload easier
- Simpler replication
- Very easy to setup with the instance manager or mysqld_multi
- mysql.com init scripts supports the instance manager
Asynchronous data loading

• Updating counts? Loading logs?

• Don’t talk directly to the database, send updates through Spread (or whatever) to a daemon loading data

• Don’t update for each request
  ```sql
  update counts set count=count+1 where id=37
  ```

• Aggregate 1000 records or 2 minutes data and do fewer database changes
  ```sql
  update counts set count=count+42 where id=37
  ```

• Being disconnected from the DB will let the frontend keep running if the DB is down!
Preload, -dump and -process

• Let the servers do as much as possible without touching the database directly

• Data structures in memory – ultimate cache!

• Dump never changing data structures to JS files for the client to cache

• Dump smaller read-only often accessed data sets to SQLite or BerkeleyDB and rsync to each webserver (or use NFS, but...)

• Or a MySQL replica on each webserver
Stored Procedures
Dangerous

- Not horizontal
- Work in the database server bad (unless it’s read-only and replicated)
- Work on one of the scalable web fronts good
- Only do stored procedures if they save the database work (network-io work > SP work)
Reconsider Persistent DB Connections

- DB connection = thread = memory
- With partitioning all httpd processes talk to all DBs
- With lots of caching you might not need the main database that often
- MySQL connections are fast
- Always use persistent connections with Oracle!
  - Commercial connection pooling products
InnoDB configuration

- `innodb_file_per_table`
  Splits your innodb data into a file per table instead of one big annoying file

- Makes optimize table `table` clear unused space

- `innodb_buffer_pool_size=($MEM*0.80)`

- `innodb_flush_log_at_trx_commit` setting

- `innodb_log_file_size`

- `transaction-isolation = READ-COMMITTED`
Store Large Binary Objects
(aka how to store images)

• Meta-data table (name, size, ...)

• Store images either in the file system
  • meta data says “server ‘123’, filename ‘abc’”
  • Replication issues! (mogilefs, clustered NFS, ...)

• **OR** store images in other (MyISAM) tables
  • Split data up so each table don’t get bigger than ~4GB

• Include “last modified date” in meta data

• Include it in your URLs to optimize caching (squid!)
  (/images/$timestamp/$id.jpg)
Random Application Notes

- Everything is Unicode, please!
- (DBD::mysql ... oops)
- Make everything use UTC – it’ll never be easier to change your app than now
- My new favorite feature:
  - Make MySQL picky about bad input!
  - `SET sql_mode = 'STRICT_TRANS_TABLES'`
Don’t overwork the DB

• Databases don’t easily scale
• Don’t make the database do a ton of work
• Referential integrity is good
  • Tons of extra procedures to validate and process data maybe not so much
• Don’t be too afraid of de-normalized data – sometimes it’s worth the tradeoffs (call them summary tables and the DBAs won’t notice)
Sessions

“The key to be stateless”

or

“What goes where”
Evil Session

Web/application server
with local
Session store

Cookie: session_id=12345

What’s wrong with this?

12345 => {
    user =>
       { username => 'joe',
         email => 'joe@example.com',
         id => 987,
       },
    shopping_cart => { ... },
    last_viewed_items => { ... },
    background_color => 'blue',
},
12346 => { ... },
....
Evil Session

Cookie: session_id=12345

Easy to guess cookie id

Web/application server with local Session store

Saving state on one server!

Duplicate data from a DB table

Big blob of junk!

12345 => {
    user => {
        username => 'joe',
        email => 'joe@example.com',
        id => 987,
    },
    shopping_cart => { ... },
    last_viewed_items => { ... },
    background_color => 'blue',
},
12346 => { ... },
....

What’s wrong with this?
Good Session!

- Stateless web server!
- Important data in a database
- Individual expiration on session objects
- Small data items in cookies

Database(s)

Users
987 =>
{ username => 'joe',
  email => 'joe@example.com',
},
...
Shopping Carts
...

memcached cache

seh568fzkj5k09z =>
{ last_viewed_items => {...},
  ... other "junk" ...
},
....

Cookie: sid=seh568fzkj5k09z;
user=987-65abc;
bg_color=blue;
cart=...;

Web/application server
Safe cookies

- Worried about manipulated cookies?
- Use checksums and timestamps to validate them!
  - cookie=1/value/1123157440/ABCD1234
  - cookie=1/user::987/cart::943/ts::1123.../EFGH9876
  - cookie=$cookie_format_version
    /$key::$value[/$key::$value]
    /ts::$timestamp
    /$md5

- Encrypt them if you must (rarely worth the trouble and CPU cycles)
Use your resources wisely

don’t implode when things run warm
Resource management

- Balance how you use the hardware
- Use memory to save CPU or IO
- Balance your resource use (CPU vs RAM vs IO)
- Don’t swap memory to disk. Ever.
Do the work in parallel

- Split the work into smaller (but reasonable) pieces and run them on different boxes
- Send the sub-requests off as soon as possible, do something else and then retrieve the results

Are the horizontal lines parallel or do they slope?
Use light processes for light tasks

• Thin proxy servers or threads for “network buffers”
• Goes between the user and your heavier backend application
• httpd with mod_proxy / mod_backhand
  • perlbal
    – new & improved, now with vhost support!
• squid, pound, ...

Proxy illustration

perlbal or mod_proxy
low memory/resource usage

backends
lots of memory
db connections etc

Users
Light processes

- Save memory and database connections
- This works spectacularly well. Really!
- Can also serve static files and cache responses!
- Avoid starting your main application as root
- Load balancing
- Very important if your backend processes are “heavy”
Light processes

• Apache 2 makes it **Really Easy**

• ProxyPreserveHost On
  
  <VirtualHost *>
    ServerName combust.c2.askask.com
    ServerAlias *.c2.askask.com
    RewriteEngine on
  
  RewriteRule (.*) http://localhost:8230$1 [P]
  </VirtualHost>

• Easy to have different “backend environments” on one IP

• Backend setup (Apache 1.x)
  
  Listen 127.0.0.1:8230
  Port 80
Job queues

- Processing time too long for the user to wait?
- Can only do N jobs in parallel?
- Use queues (and an external worker process)
- AJAX can make this really spiffy
Job Queues

- Database “queue”
  - Webserver submits job
  - First available “worker” picks it up and returns the result to the queue
  - Webserver polls for status
- Other ways...
  - gearman
  - Spread
  - MQ / Java Messaging Service(?) / ...
Log http requests!

- Log slow http transactions to a database
  time, \texttt{response\_time}, uri, remote\_ip, user\_agent, request\_args, user, svn\_branch\_revision, log\_reason (a “SET” column), ...

- Log 2\% of all requests!

- Log all 4xx and 5xx requests

- Great for statistical analysis!
  - Which requests are slower
  - Is the site getting faster or slower?

- \texttt{Time::HiRes} in Perl, microseconds from \texttt{gettimeofday} system call
Get good deals on servers

- **Silicon Mechanics**
  http://www.siliconmechanics.com/
- Server vendor of LiveJournal and lots others
- Small, but not too small
remember

Think Horizontal!
Hiring!

- Contractors and dedicated moonlighters!
- Help me with $client_project ($$)
- Help me with $super_secret_startup (fun!)
- Perl / MySQL
- Javascript/AJAX
- ask@develooper.com
  (resume in text or pdf, code samples)
Thanks!

• Direct and indirect help from ...
  • Cal Henderson, Flickr
  • Brad Fitzpatrick, LiveJournal
  • Kevin Scaldeferri, Overture Yahoo!
  • Perrin Harkins, Plus Three
  • Tim Bunce
  • David Wheeler, Tom Metro
Questions?

Thank you!

More questions? Need consulting?
ask@perl.org
ask@develooper.com

http://develooper.com/talks/