



PERCONA  
Performance Consulting Experts

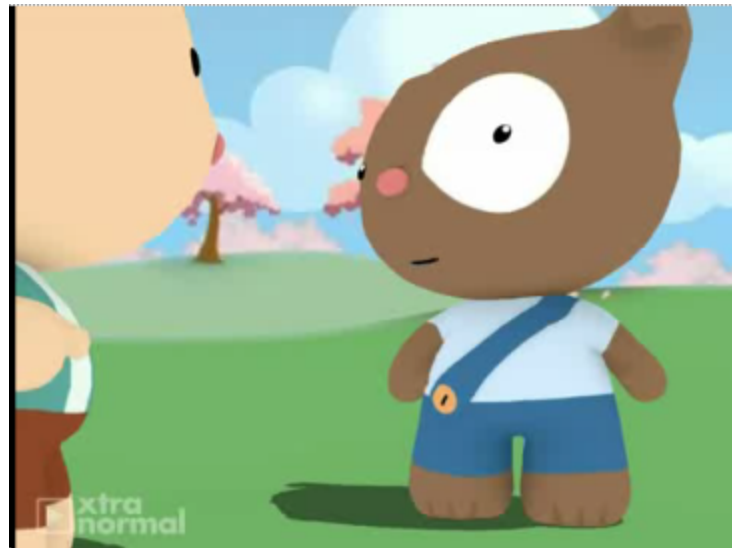
---

# Scaling Without Sharding

Baron Schwartz  
Percona Inc  
Surge 2010

# Web Scale!!!!

<http://www.xtranormal.com/watch/6995033/>



# A Sharding Thought Experiment

- 64 shards per proxy<sup>[1]</sup>
- 1 TB of data storage per node
  - Including room for compaction, nodes need 2 TB of disk
- With a single proxy, maximum 64 TB of data
  - 128 or perhaps 192 server nodes, depending on the level of redundancy required by the system
- Continued on the next slide...

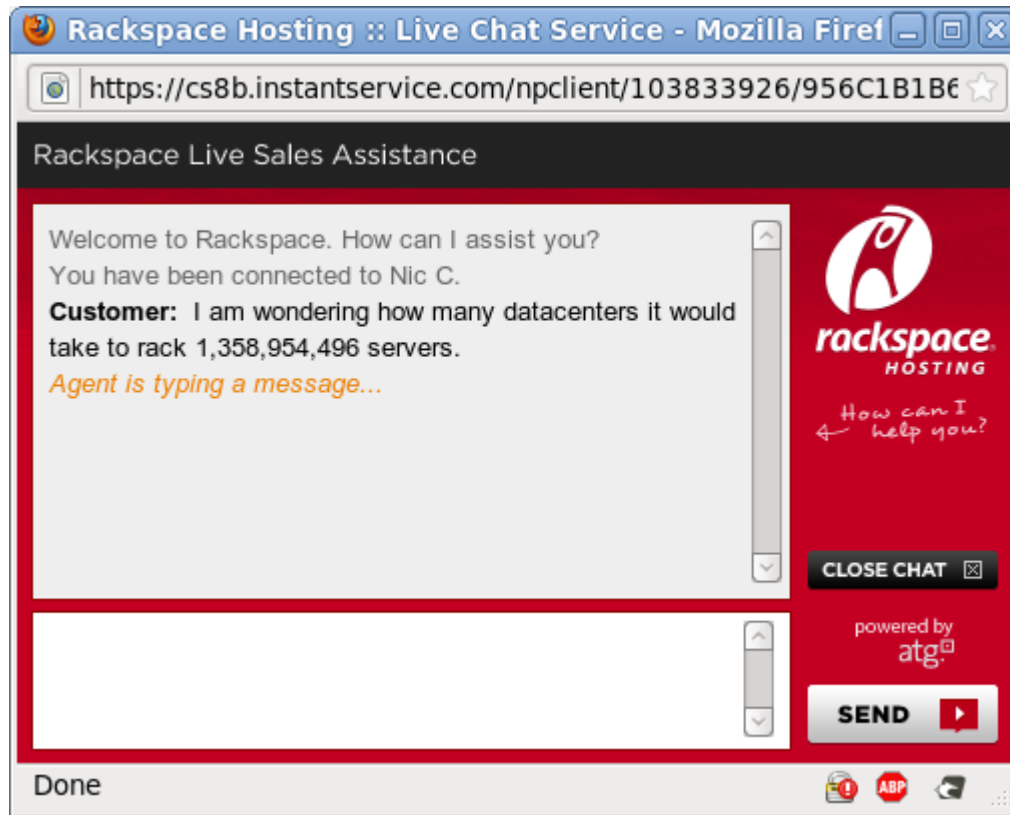
[1] This “thought experiment” is quoted almost word-for-word from a recently published book on a specific database, which I will not mention because it doesn't matter. The thought process is what is interesting for this discussion.

# Going to Web Scale

- Basic arithmetic: with three layers of proxies
  - 262 petabytes on thousands of machines.
- Conservative estimate: latency per layer ~100ms
  - Overall response times of 300 ms
  - Even without performance tuning!
- We should be able to manage queries over exabyte datasets in less than a second.
- *Now, what's wrong with this theoretical process of scaling to handle exabyte datasets?*

# Web-Scale Reality

- $192^4 = 1,358,954,496$  servers<sup>[1]</sup>



[1] My hasty math was wrong when I gave this talk. It should be  $(64^4)*3$ , which is more than 50 million servers. Note to self: check the math more carefully.

# Web-Scale Reality

Rackspace Hosting :: Live Chat Service - Mozilla Firefox

https://cs8b.instantsservice.com/npclient/103833926/956C1B1B6

Rackspace Live Sales Assistance

Welcome to Rackspace. How can I assist you?  
You have been connected to Nic C.

**Customer:** I am wondering how many datacenters it would take to rack 1,358,954,496 servers.

**Nic C:** That would depend upon the type of server..

**Nic C:** Is this for a current project that you are working on?

**rackspace**  
HOSTING

How can I help you?

CLOSE CHAT

powered by atg

SEND

Done

# Beating the Dead Horse

---

- Remember,
  - “1TB... nodes need 2 TB of disk”
  - So for 1 exabyte of storage, we need 1 exabyte of idle storage.
  - That's roughly 40 megawatts of idle storage.
  - Surely there is a better way to architect this system?
- Many other things are questionable.
  - Network bandwidth at upper-level proxies?
  - Is the estimated latency really achievable?
  - And so on. Some solvable, some not.

# Shard Early, Shard Often?

---

Where are we today? Really?

# Sharding was required when...

---

- MySQL 5.0 with InnoDB that couldn't scale.
- SSDs were only seen in *Popular Mechanics*.

# What is Scaling?

---

# What is Scaling?

---

The ability to do more of X  
while ensuring that Y is acceptable

*The ability to add capacity to a system.*

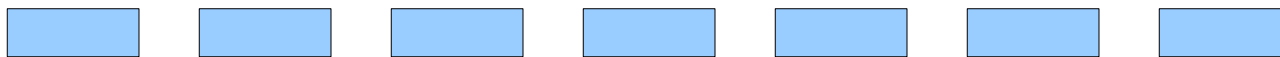
# 3 Fundamental Techniques



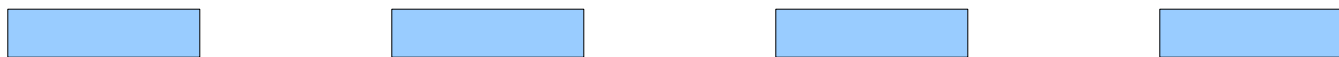
- Add more capacity to do X



- Make the system do X faster



- Stop doing so much X



# In other words, scaling is...

---

---

Scaling is about performance.

# Performance Is...

---

- Response time (time per task)
- Throughput (tasks per time)

# Digression!

The art of optimizing performance™

- 1) Measure<sup>[1]</sup> where X spends its time
- 2) Reduce time consumption

[1] cf. Neil Gunther: “All measurements are wrong by definition.”

# Scaling Directions

---

Scale-out -vs- Scale-up

# Scaling Directions

---

- Scale-out -vs- Scale-up is not very helpful.
- Understanding the whole application is helpful.

---

Let's think about the whole application.

# What makes scaling easy?

---

- If scaling is about performance, and
- Performance is about tasks and time, then
- Generally, what makes a task consume time?
  - It is utilizing a resource.
  - It is waiting for a response.
  - It is waiting for synchronization.
- What do these activities have in common?

# Scaling is easy when...

---

- Components are decoupled from each other.
- Components are stateless.
- Components operate asynchronously.

# App Servers are Easy to Scale

---

- App servers are easy to scale because they:
  - Are usually decoupled.
  - Delegate inter-request statefulness.
  - Delegate inter-process synchronization.
- This means that...

# Delegation is Key

---

App servers are easy to scale because of delegation.

Generally, app servers delegate their writes.

They have to delegate this work somewhere.

Database servers are usually on the receiving end.

# ACID Database Servers Are...

---

- Stateful
- Synchronous
- (Internally) centralized, i.e. not decoupled
- Therefore,
  - Hard to optimize for performance
  - Hard to scale

# Now, how about scaling?

---

- So much for “motivating the discussion”
- How do we actually scale something? *Anything?*

# Back to performance

---

- Scalability = Performance = Tasks and Time
- Two fundamental tasks to perform with data:
  - Read
  - Write
- Each presents different opportunities.

# The Nature of Reads

---

- Reads are inherently synchronous.
- Reads are cachable.
- Reads can be combined.
- Reads require 100% of necessary resources.
- Reads can be done anywhere the data is.

# The Nature of Writes

---

- Writes can be asynchronous (fire-and-forget).
- Writes can be buffered (delegated/queued).
- Writes can be combined.
- Writes can be done with partial resources.
- Writes must be done everywhere the data is.

# Two Fundamental Patterns

---

- Get more out of one server
  - Advantages: too many to mention
  - Disadvantages: ???
- Use multiple servers
  - Advantages: when one server isn't enough
  - Disadvantages: too many to mention

# Single Server Method #1

---

- Add more resources.
  - Storage (Disk, RAM)
  - CPU
  - Communication (Network, Bus)
- This is hard in the cloud.
- Most (non-cloud) deployments can go *very far*

# Single Server Method #2

---

- Avoid using slow resources
  - Disk (yes, even SSD)
- Primarily, this means *fit the working set in RAM*
  - Archiving, purging
    - Locality of hot data
  - Summarizing
  - Compression
  - Choice of data types
  - Schema design and physical layout
    - Clustered indexes

# When One Isn't Enough

---

- When demand exceeds resources...
- What kind of demand?
  - Writes?
  - Reads?
- Options:
  - Duplication: Make multiple copies of the data
  - Sharding: Partition the data

# Strategy: Duplicated Data

---

- Caching
- Indexes
- Replication
- CDNs
- Materialized views

# Strategy: Duplicated Data

---

- Useful for increasing read capacity
  - Because reads can be done anywhere the data is.

# Strategy: Duplicated Data

---

- Hard problems:
  - Expiring stale data
  - Knowing whether data is consistent
  - Updating the copies when the master copy changes
- Drawback:
  - Each copy consumes resources
  - Hard to exploit locality of reference
  - Works best when application is mostly reads
    - Because writes must be done everywhere

# Strategy: Sharding

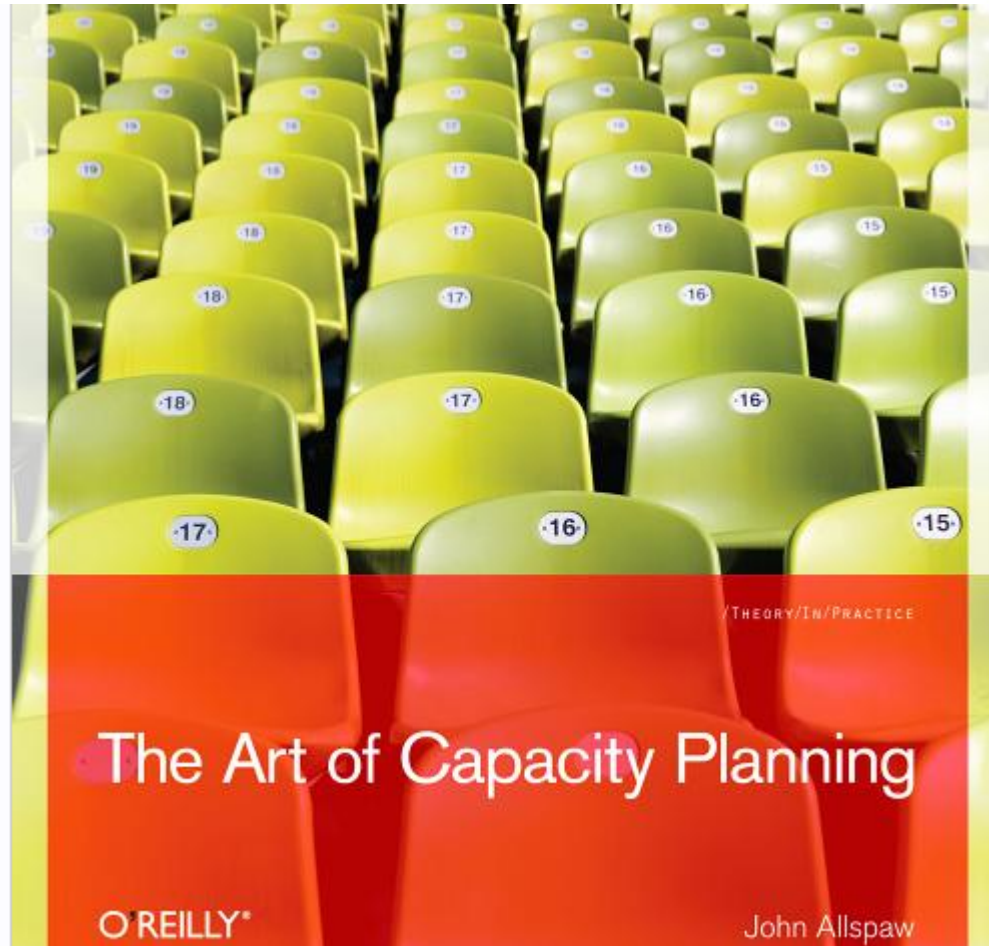
- Sharding should be your last resort.
- Forced when *write demand exceeds capacity*.
- Not as much of a problem as it used to be.
- SSDs change things somewhat.
- But:
  - MySQL replication is single-threaded.
  - Replica can't keep up with the primary.
  - Cloud platforms don't have SSD performance.

# Avoiding Sharding

- Buffer writes.
  - Buffering permits write-combining (click counting?)
  - Lets the caller fire-and-forget (queue).
- Pre-process writes.
  - Don't store and aggregate in the database.
  - Store outside the database, aggregate, then load in.
  - Aggregate on a replica, load results to the primary.
- Combine pushing and pulling.
  - Push (synchronous) to a buffer or queue.
  - Asynchronously pull and process.

# Avoiding Sharding

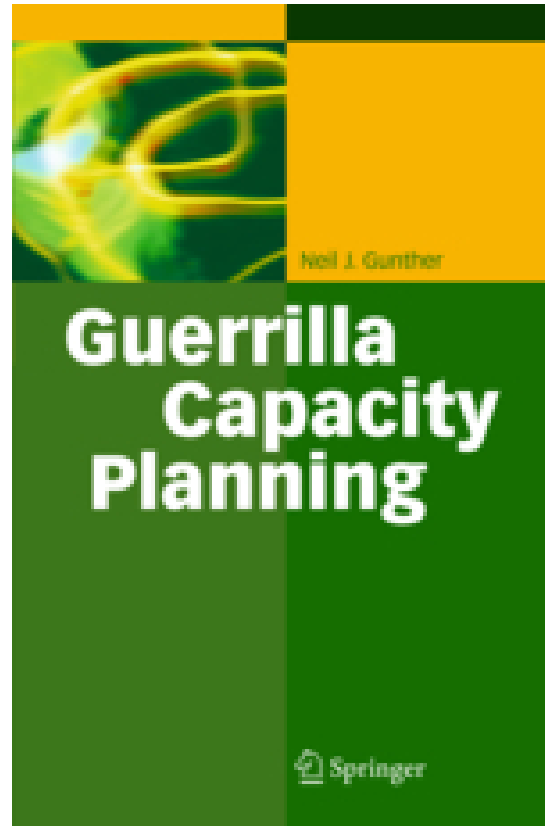
- Combine buffering with caching
  - Send the changes to a buffer or queue
  - Duplicate the changes to the cache
  - Hit the cache on reads
- Delay full updates
  - Just log the change instead of performing it
  - Later, process the result, doing 100% of the work
- Hit the hot data last
  - Avoid the contended data until the last minute
  - Then modify it and commit ASAP



Theo Schlossnagle

## Scalable Internet Architectures





*Optimization, Backups,  
Replication, and more*

**2nd Edition**  
*Covers Version 5.1*



# High Performance MySQL

**O'REILLY®**

*Baron Schwartz, Peter Zaitsev,  
Vadim Tkachenko, Jeremy D. Zawodny,  
Arjen Lentz & Derek J. Balling*

# Questions & Comments

---

<first name>@percona.com