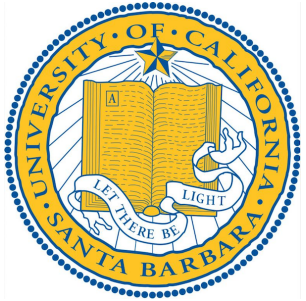
CS 290B

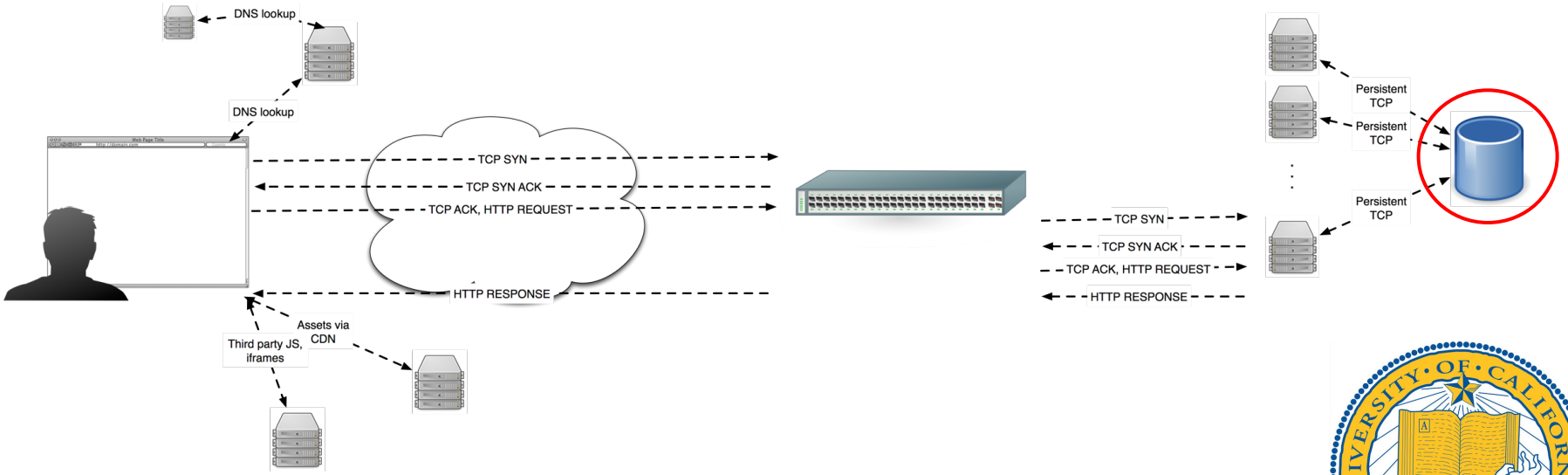
Scalable Internet Services

Andrew Mutz

October 30, 2014



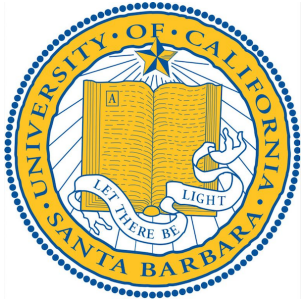
For Today


DNS lookup
DNS lookup
TCP SYN
TCP SYN ACK
TCP ACK, HTTP REQUEST
HTTP RESPONSE
Assets via CDN
Third party JS, iframes
TCP SYN
TCP SYN ACK
TCP ACK, HTTP REQUEST
HTTP RESPONSE
Persistent TCP
Persistent TCP
Persistent TCP

Motivation

You've got your application running on EC2. It is becoming increasingly popular and performance is degrading.

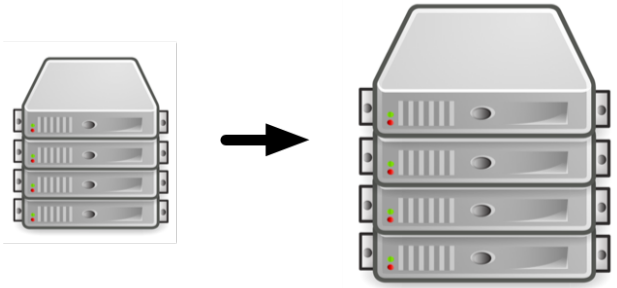
What do you do?



Motivation

You've got your application running on EC2. It is becoming increasingly popular and performance is degrading.

What do you do?



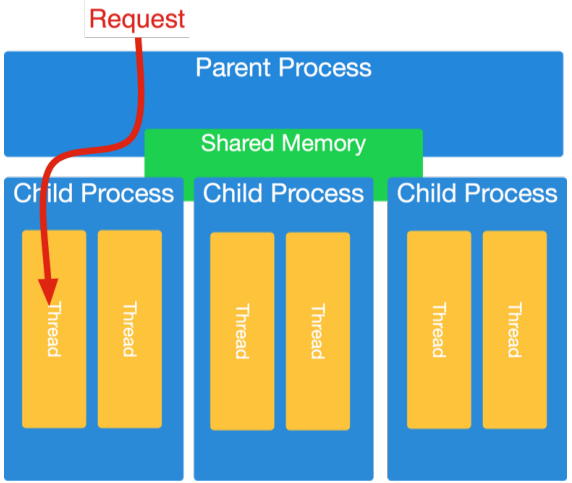
You've increased instance sizes, and that buys some time. But as popularity grows, there are no larger instances to buy.

Motivation

You've got your application running on EC2. It is becoming increasingly popular and performance is degrading.

What do you do?

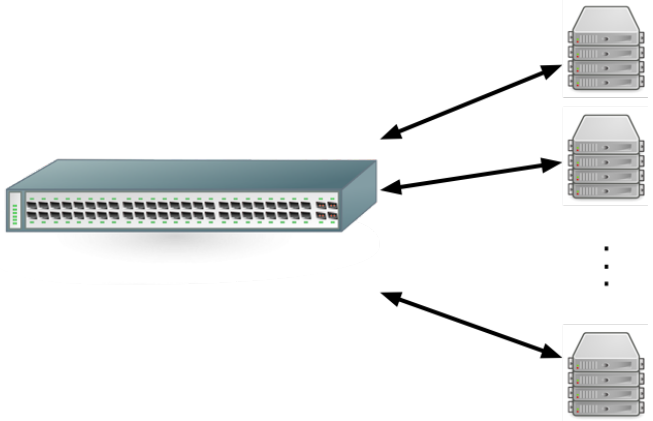
You've deployed application servers that can serve many requests simultaneously, and that helped. But as popularity grows, it's not enough.



Motivation

You've got your application running on EC2. It is becoming increasingly popular and performance is degrading.

What do you do?

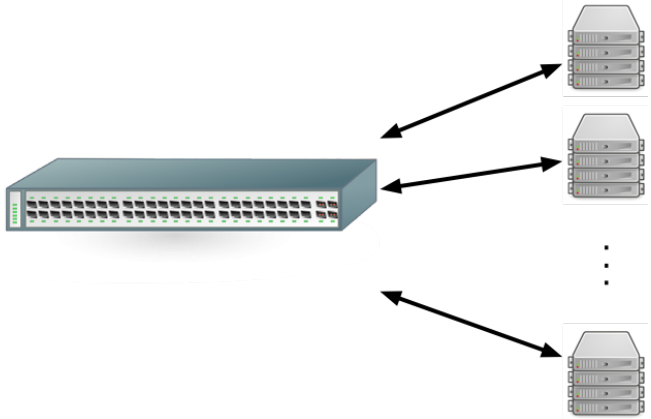


You've set up a load balancer and spread load across many servers. But as popularity grows, it's not enough.

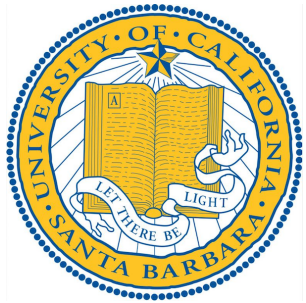
Motivation

You've got your application running on EC2. It is becoming increasingly popular and performance is degrading.

What do you do?



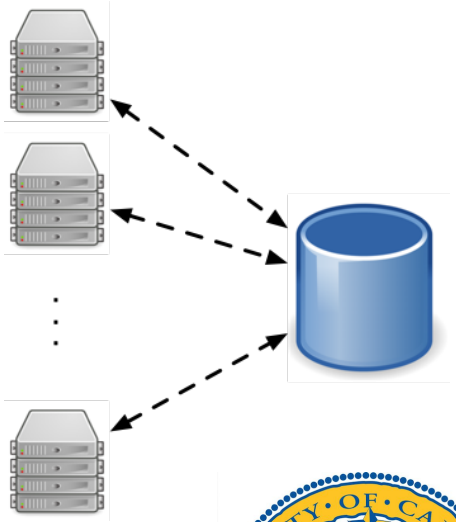
You've set up HTTP caching correctly so unnecessary requests never happen. You're caching heavyweight database operations. But as popularity grows, it's not enough.



Motivation

You've got your application running on EC2. It is becoming increasingly popular and performance is degrading.

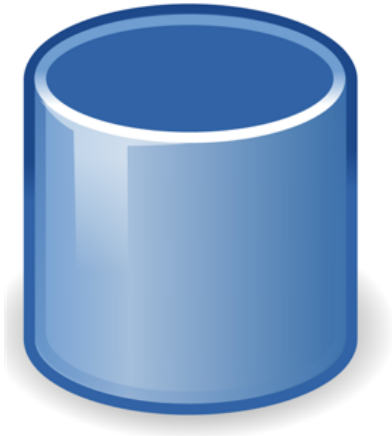
What do you do?



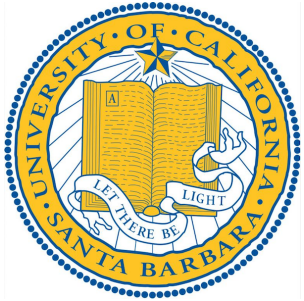
As you keep adding application servers, your database struggling to keep up. You've used EXPLAIN to detect missing indices and add them. You've detected slow queries and fixed them. But as popularity grows, it's not enough.

Motivation

Your relational database is still on one machine and that machine is having trouble keeping up with increased load.

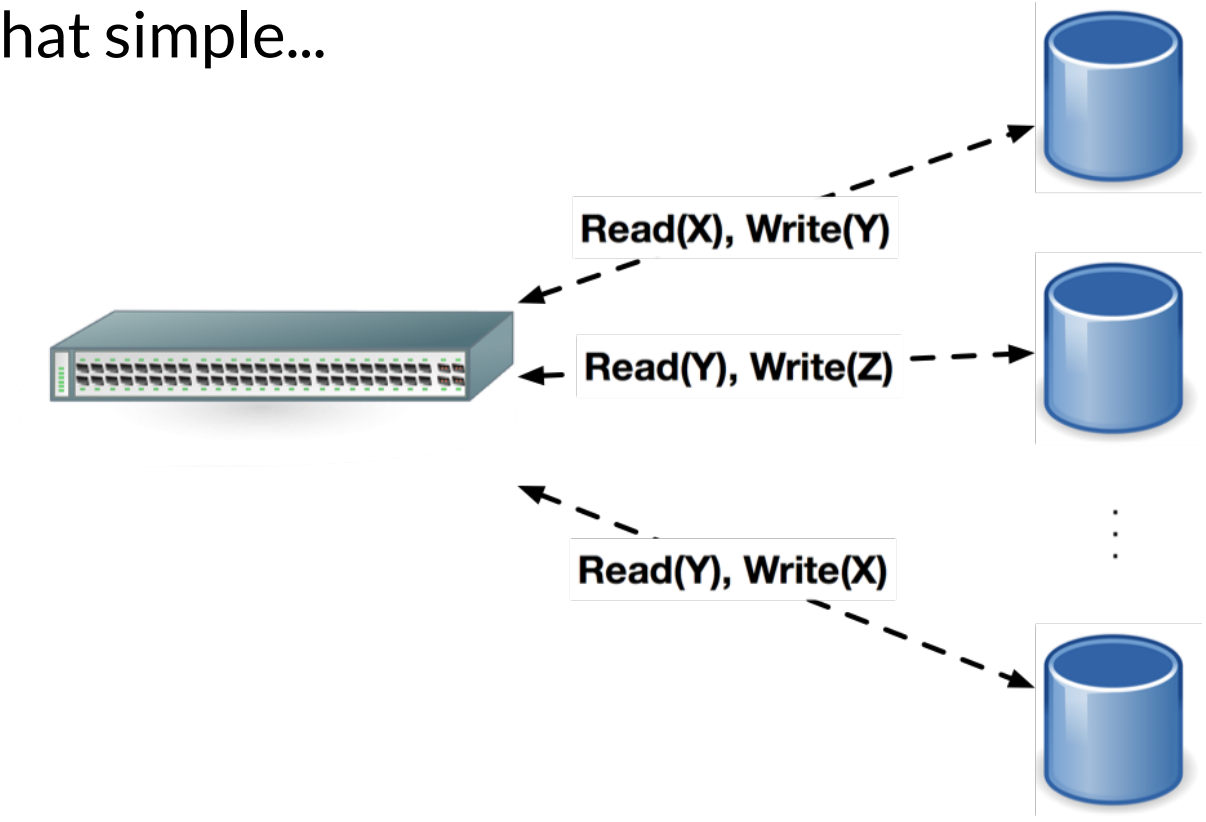


Can we just put a load balancer in front of the database and spread load across many databases?



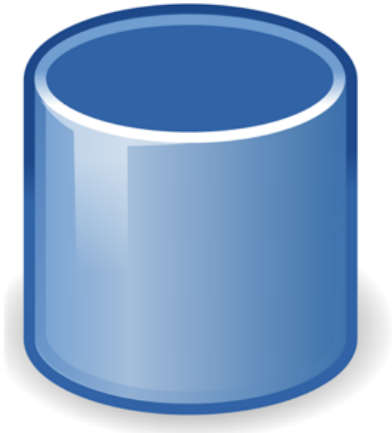
Motivation

It’s not that simple...



Motivation

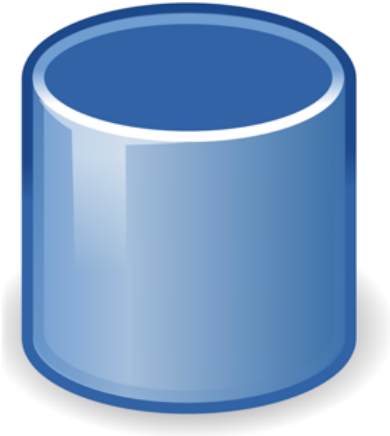
Relational databases are hard to scale.



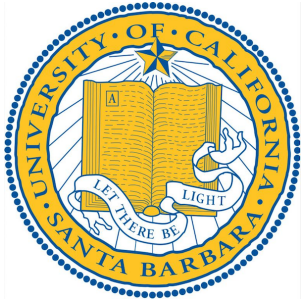
In future lectures, we will discuss scaling the data layer using non-relational data stores.

Today we will talk about what we **can** do to scale with relational databases.

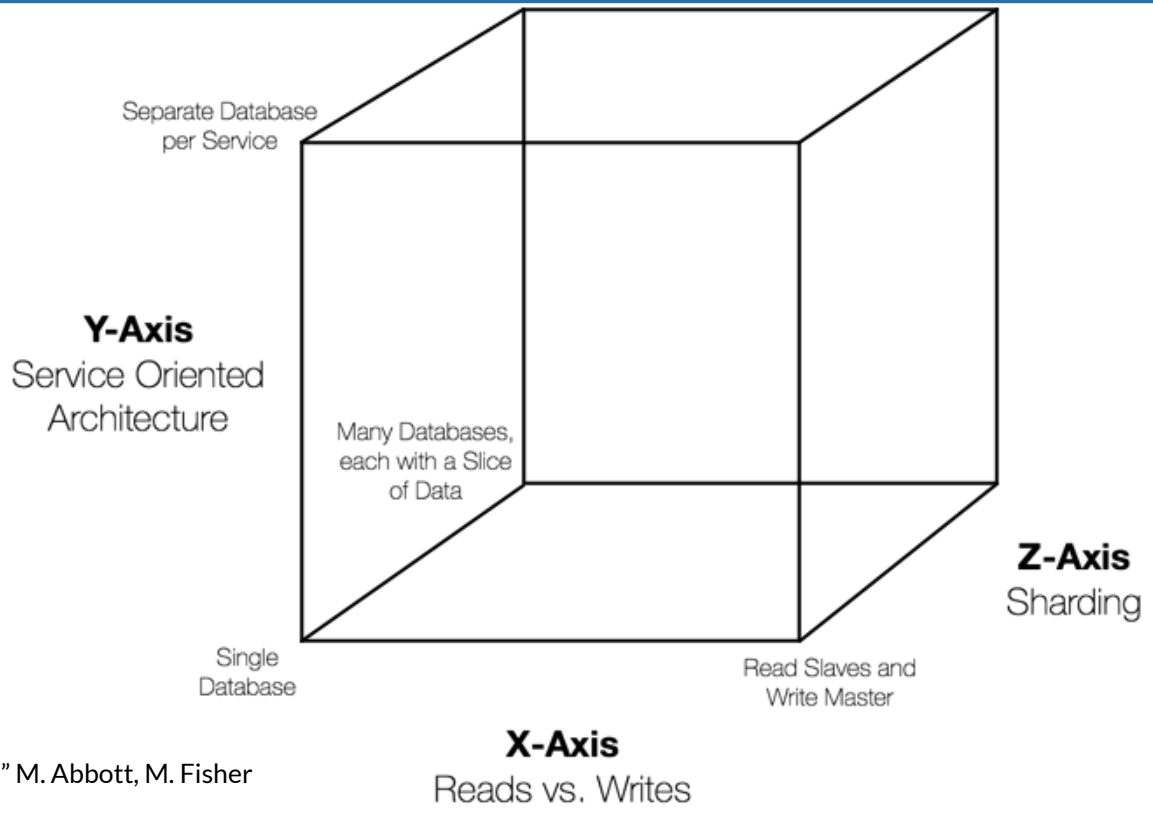
Today's Agenda



- Sharding
- Service Oriented Architectures
- Distinguishing Reads from Writes



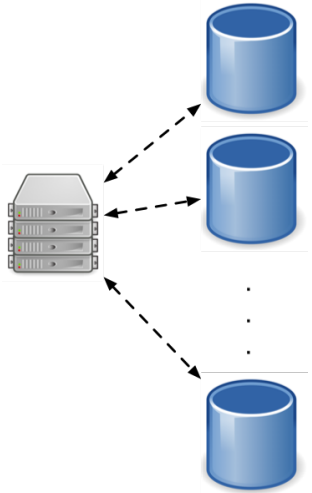
AKF Cube



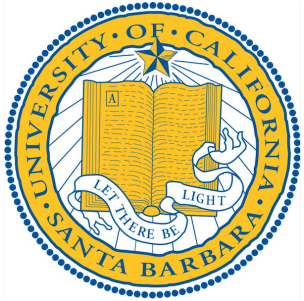
Source: “Scalability Rules,” M. Abbott, M. Fisher

Sharding

Sharding



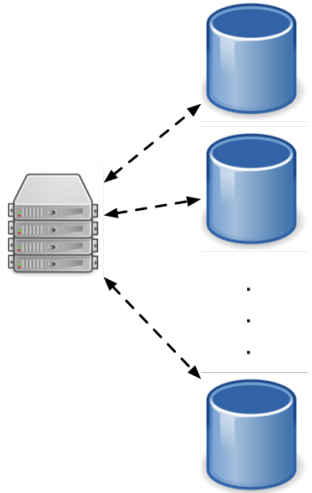
- Find partitions that cleanly divide your dataset, and separate your data into separate databases.
  - Split out similar data across RDBMSes.
- You take something you know about a request, and use that to choose which database to talk to.
  - Examples: customer, geography
- Sometimes referred to as Podding



Sharding

Sharding

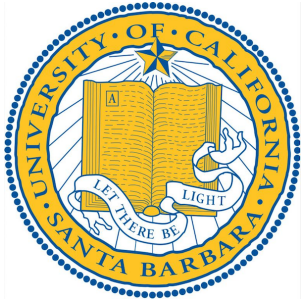
- Ideally, your partitions increase as usage of your application increases.
- Example:
  - If each customer's data can be partitioned from the rest, then doubling the number of customers doubles the number of clean partitions.



Sharding

Simple Example:

- Gmail!
- The data that represents my email needs no relations to the data representing other people's email
- When a request arrives, we apply some mapping function to determine which database it belongs to
- We can cleanly partition users into separate data stores
    - Scales beautifully as number of users increases

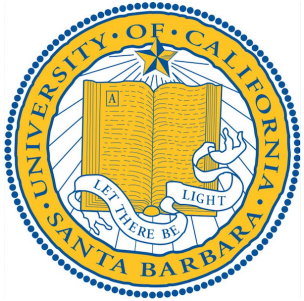


Sharding

Harder Example: Our lab app.

- Users can create and view communities
- Those communities have lists of submitted links
- Each submission has a tree of comments

How should we divide up (shard) this application?



Sharding

By User?

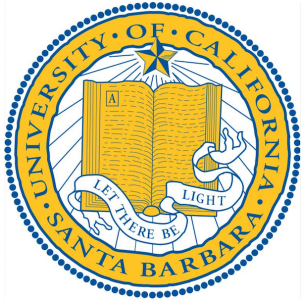
- Would be awkward, since logged in users will want to see submissions and comments by other users

By Submission?

- Users should be able to view many submissions when looking at a community

By Community?

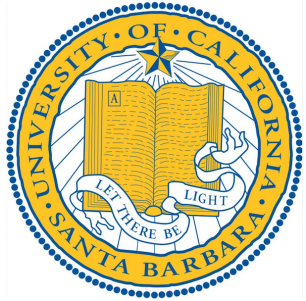
- Maybe...



Sharding

How would this work?

- Upon receiving a request, the app server would figure out which database it needed to speak to in order to serve this request
  - http://gardening.labapp.com/
  - http://labapp.com/gardening/
- After connecting to the correct database, it would issue SQL queries as normal and have everything it needed.



Sharding

Some parts of our user interface would work with community-based sharding

Bboetest Communities New Community

Title: Use of GOTO considered harmful

Url: http://www.google.com

Community: Programming

Comments

Edit | Back

Log In | Sign Up

Bboetest Communities New Community

Name: Programming

Submissions

Title	Url	
Use of GOTO considered harmful	http://www.google.com	ShowEditDestroy
A Mathematical Theory of Communication	http://bell-labs.com	ShowEditDestroy

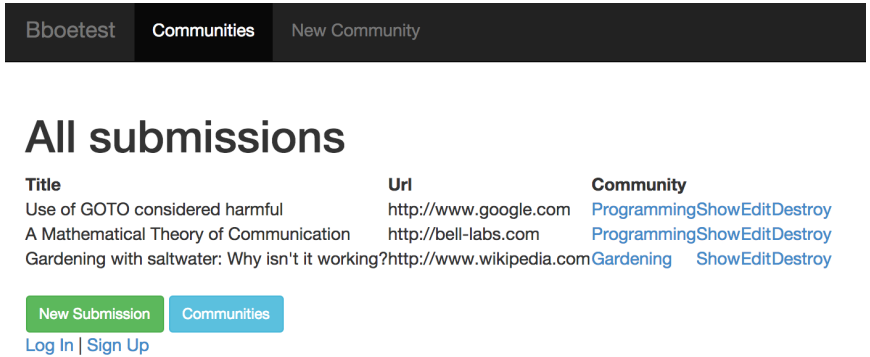
Edit Back

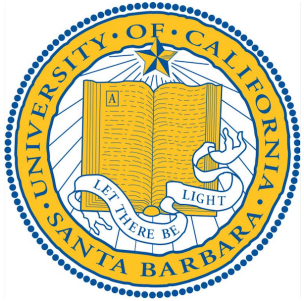
Log In | Sign Up

Sharding

Others would be more difficult:

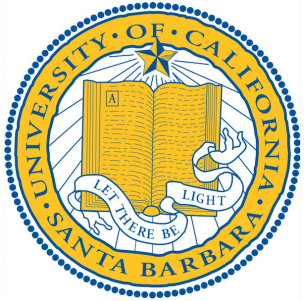
- Global views of all submissions
- Anything displaying a user's behavior across communities





Sharding

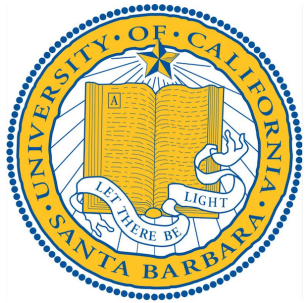
Solutions?



Sharding

Solutions:

- Modify the user interface so the difficult to shard page doesn't need to exist.
  - Maybe a semi-static list of communities and you need to dig down into each to see what is submitted.
- Construct that page using other methods
  - Separate application exists to answer specific questions about user management



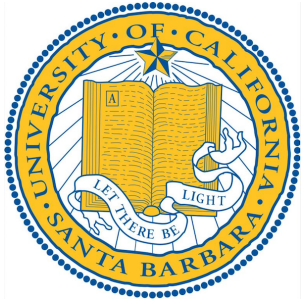
Sharding

Sharding in Rails

- Nothing built in, but the gem “Octopus” performs this well (https://github.com/tchandy/octopus)

```
class ApplicationController < ActionController::Base
  around_filter :select_shard

  def select_shard(&block)
    Octopus.using(:brazil, &block)
  end
end
```

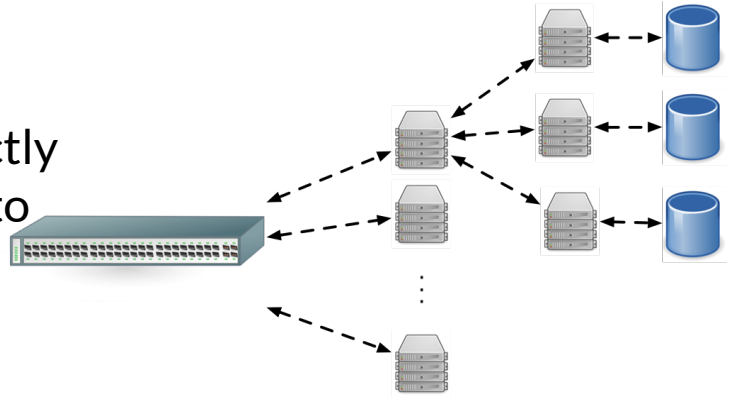


Sharding

- Strengths
  - If you genuinely have zero relations across shards, this scaling path is very clean
  - Works best when partitions grow with usage.
- Weaknesses:
  - Can inhibit feature development
    - Your app might be perfectly shardable today, but future features may change that.
  - Not easy to retroactively add to an application.
  - Transactions across shards don't happen.

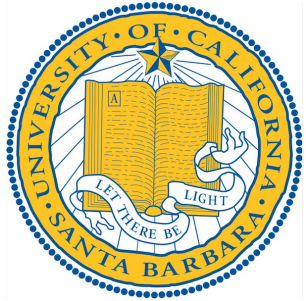
Service Oriented Architecture

Service Oriented Architecture (SOA)



- Instead of your web application speaking directly and exclusively to a single database, it speaks to other internal web applications.
- These internal applications expose an API to perform services for other applications.
- Example:
  - Internal “Users” service that we talked about in the sharding discussion.

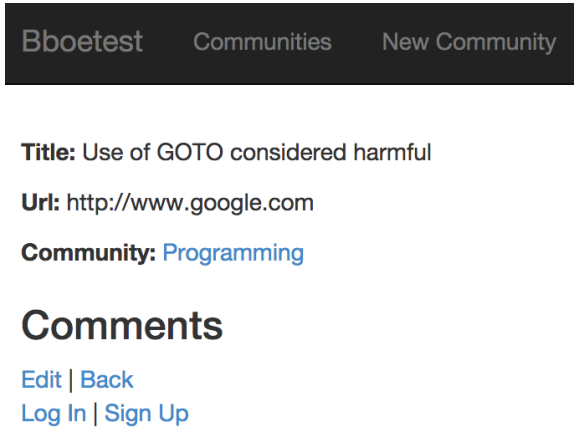
- As compared to sharding, this is splitting based on verbs instead of nouns.

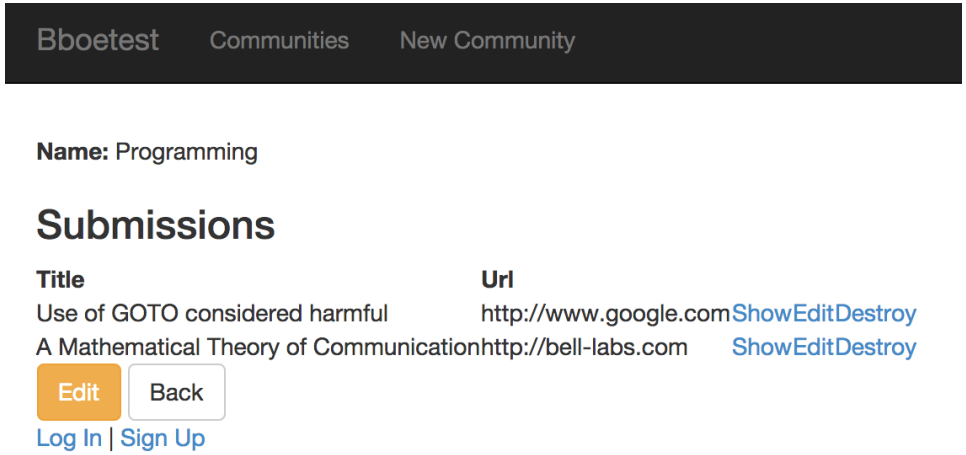


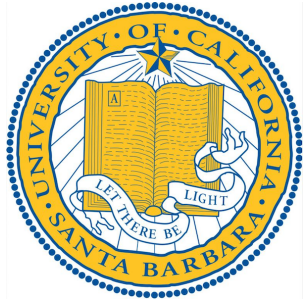
Service Oriented Architecture

SOA Example: the LabApp

How could we divide this up (via SOA)?



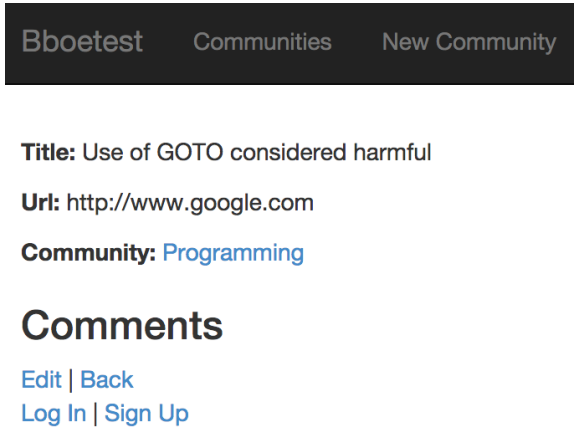


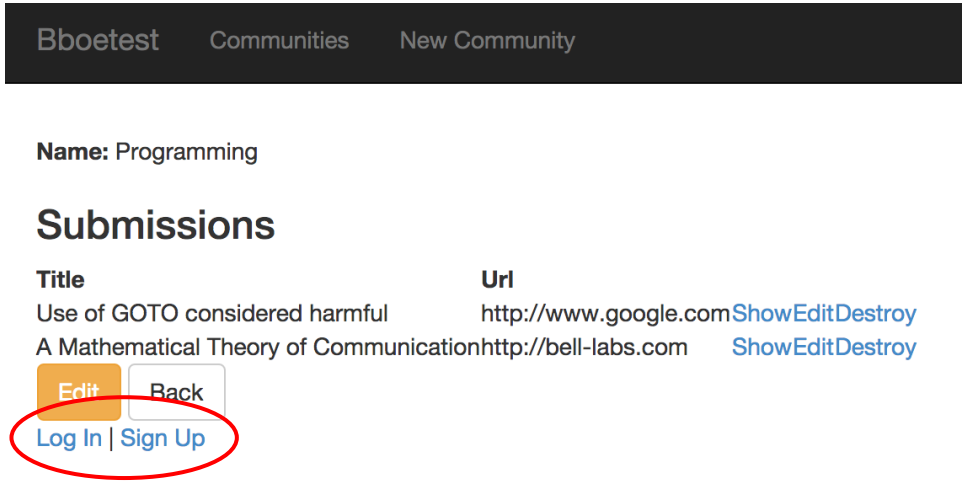


Service Oriented Architecture

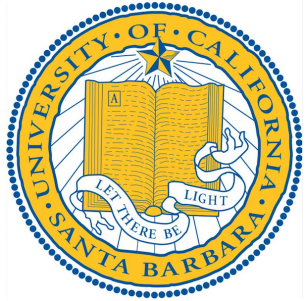
SOA Example: the LabApp

How could we divide this up (via SOA)?





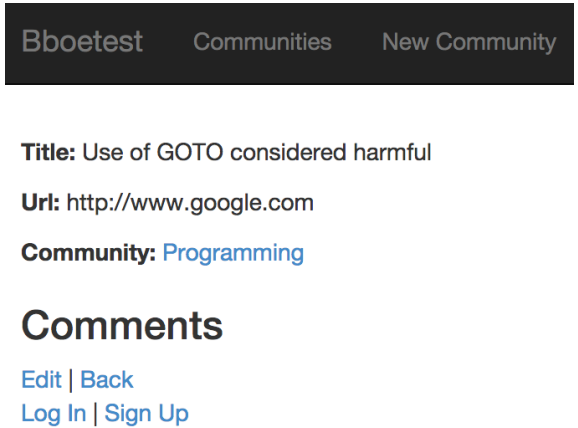
Users service keeps track of users, passwords, authentication, authorization

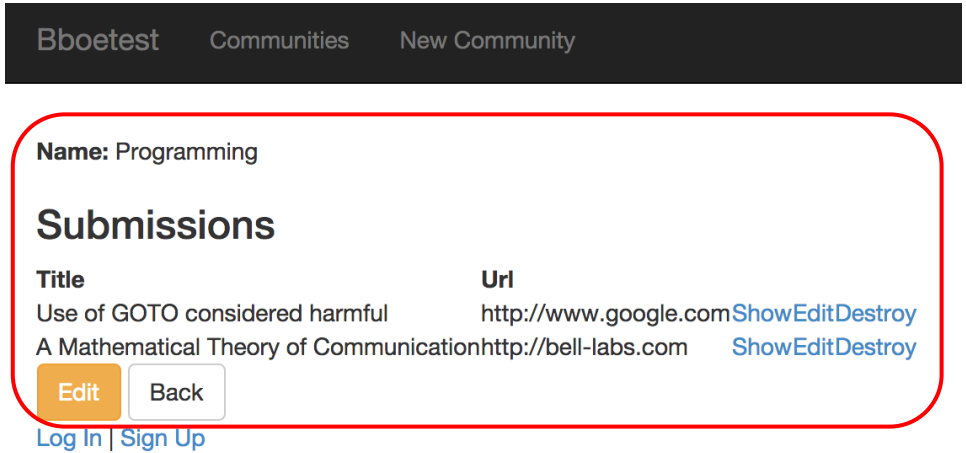


Service Oriented Architecture

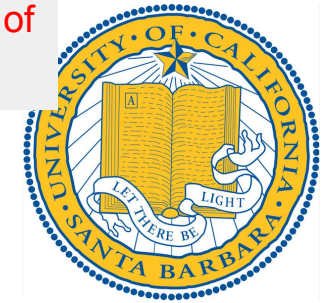
SOA Example: the LabApp

How could we divide this up (via SOA)?





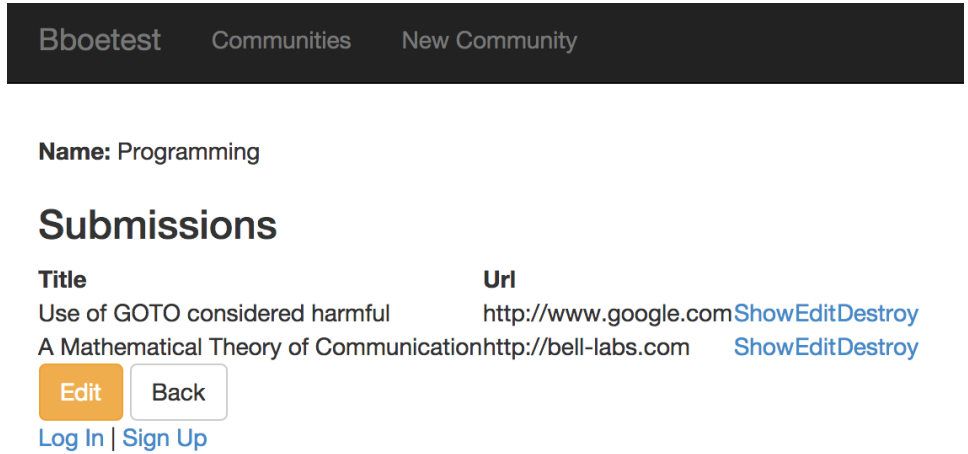
Submissions Service keeps track of Communities and Submissions

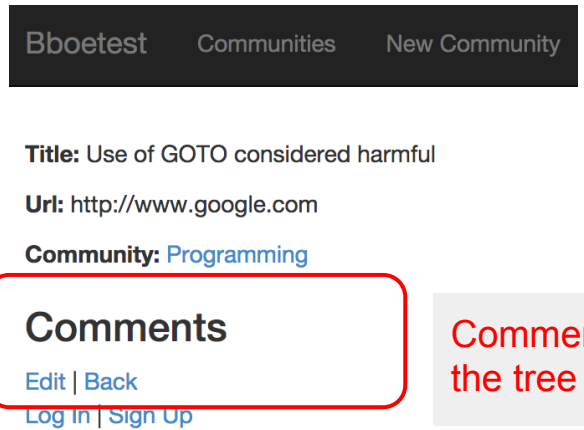


Service Oriented Architecture

SOA Example: the LabApp

How could we divide this up (via SOA)?





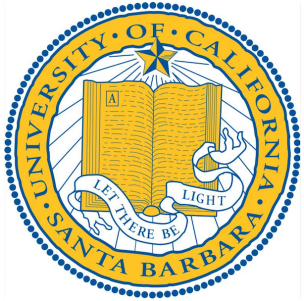
Comments Service keeps track of the tree of comments

Service Oriented Architecture

Example code change:

```
class SubmissionsController < ApplicationController
  def create
    check_user_session_in_database(cookie)
    success = write_new_submission_to_database()
    if success
      render :show
    else
      render :new
    end
  end
end
```

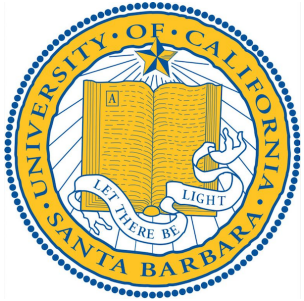
```
class SubmissionsController < ApplicationController
  def create
    user= UsersService.get_user_from_session
(cookie)
    if user.allowed_to_create_submission?
      title = params[‘title’]
      community = params[‘community’]
      success= CommunitiesService.create_submission
(
        title, community, user_id)
      if success
        render :show
      else
        render :new
      end
    end
  end
```



Service Oriented Architecture

SOA Implementation

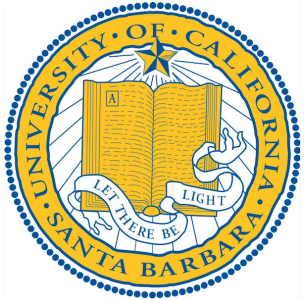
- SOA these days is commonly implemented by JSON over HTTP. RESTful APIs are common.
- Why JSON/HTTP/REST?
  - Easily constructed: Rails makes you do work to **not** have a JSON API.
  - Easily discovered: HTTP and JSON are both very readable. Documentation can be very light when the API is readable.



Service Oriented Architecture

SOA Implementation

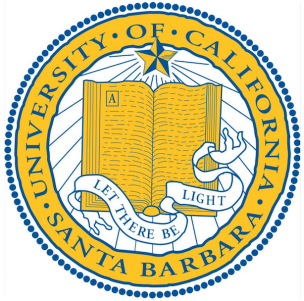
- JSON/HTTP/REST disadvantages: performance.
- For high-performance internal APIs, use
  - Google Protocol Buffers
  - Apache Thrift. (from Facebook)
- These options are generally more strongly typed (they need an encoding format) and need more documentation.



Service Oriented Architecture

Amazon famously uses this approach extensively. 2002 memo from Jeff Bezos:

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology they use.
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- **Anyone who doesn't do this will be fired. Thank you; have a nice day!**



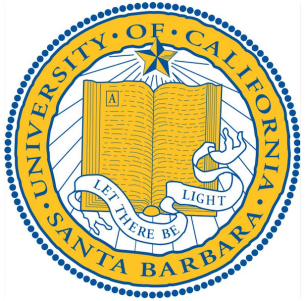
Service Oriented Architecture

Strengths

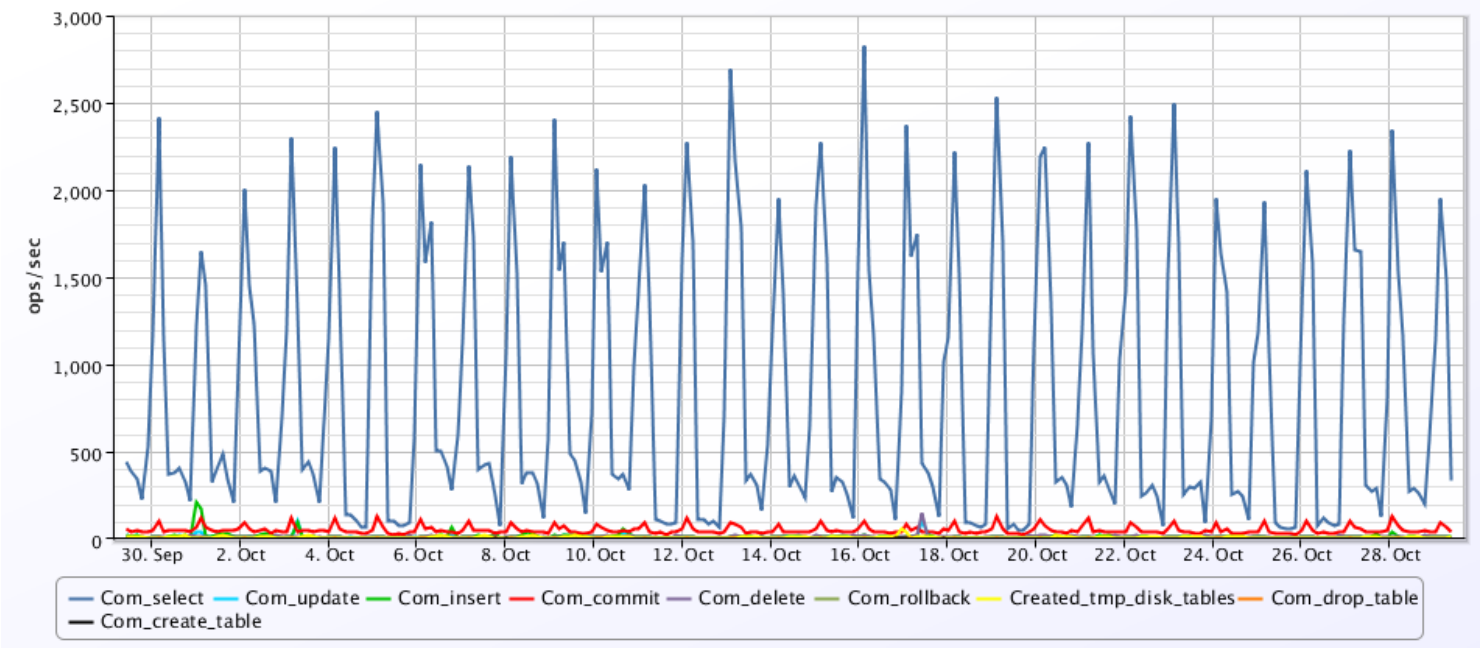
- Other benefits: small, encapsulated codebases.
- Scales well as application size scales.
- Scales well as team size scales.

Weaknesses:

- Doesn't necessarily scale as the number of users increases.
- Transactions across services don't happen.



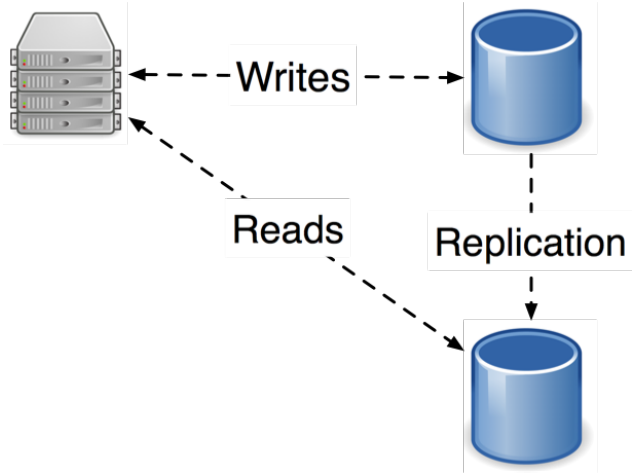
Reads vs. Writes

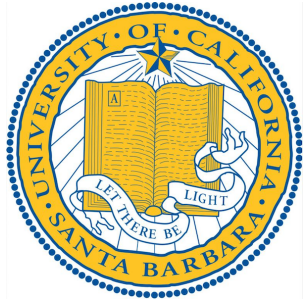


Reads vs. Writes

Observations:

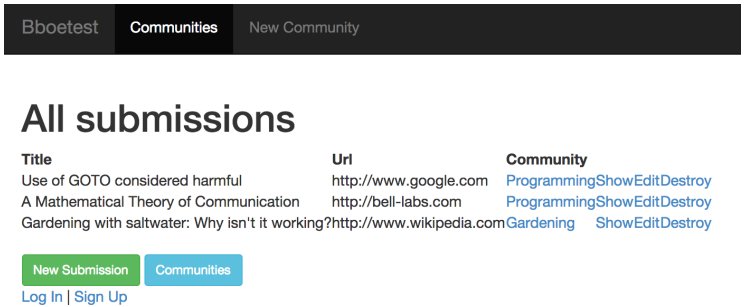
- For most web applications, reads are much more common than writes.
- Scaling out read-only copies of the database is easy.
  - If there are no writes, there can be no conflicting schedules
- Conclusion: perform reads on a different machine than your writes.

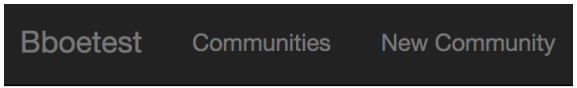




Reads vs. Writes

None of these interfaces need to write to the database...





Title: Use of GOTO considered harmful

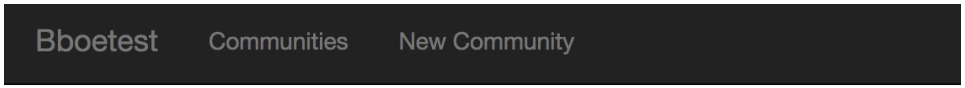
Url: http://www.google.com

Community: Programming

Comments

Edit | Back

Log In | Sign Up



Name: Programming

Submissions

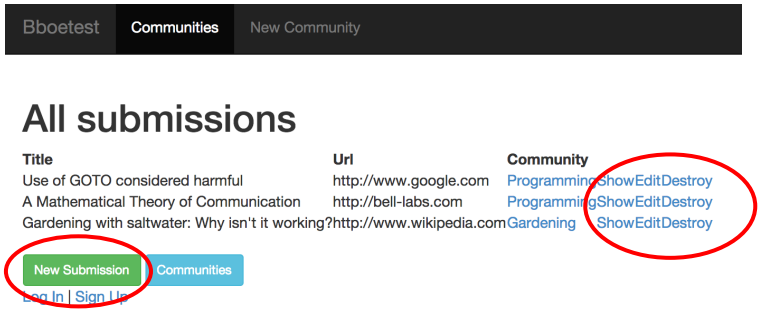
Title	Url	
Use of GOTO considered harmful	http://www.google.com	ShowEditDestroy
A Mathematical Theory of Communication	http://bell-labs.com	ShowEditDestroy

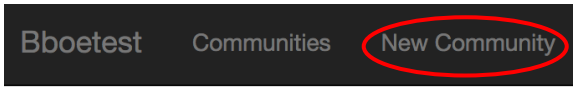
Edit Back

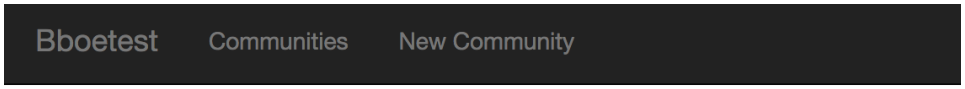
Log In | Sign Up

Reads vs. Writes

...but these would.







Title: Use of GOTO considered harmful

Url: http://www.google.com

Community: Programming

Comments

Edit | Back

Log In | Sign Up

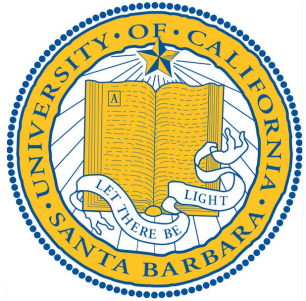
Name: Programming

Submissions

Title	Url	
Use of GOTO considered harmful	http://www.google.com	Show Edit Destroy
A Mathematical Theory of Communication	http://bell-labs.com	Show Edit Destroy

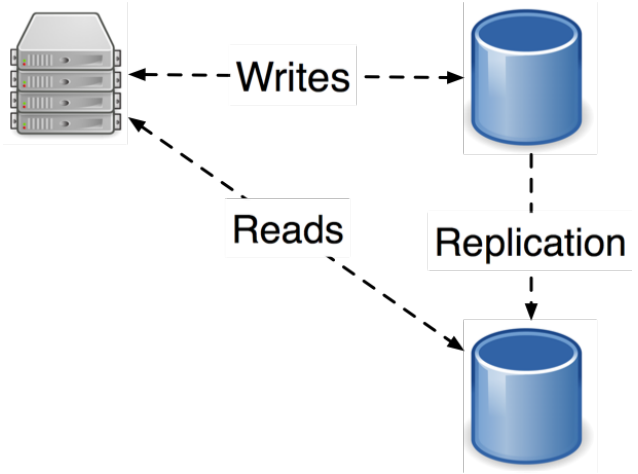
Edit Back

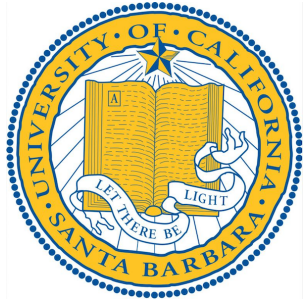
Log In | Sign Up



Reads vs. Writes

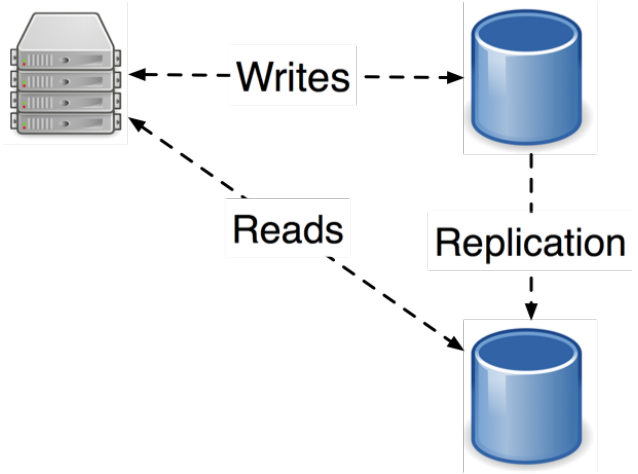
- MySQL supports this as Master/Slave replication
- Slaves can be set up to replicate to other slaves
  - This makes read-slaves effectively limitless
- Replication is asynchronous
  - Updates happen very quickly, but not immediately.

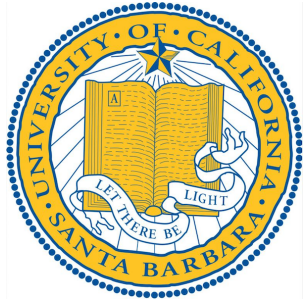




Reads vs. Writes

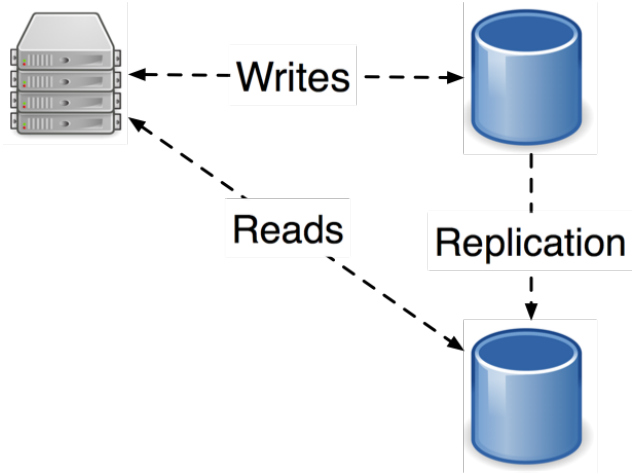
- Replication is asynchronous
  - This means writes will eventually appear on the slave, but not immediately.
  - Without this, it would be difficult/slow to scale.
- **What does asynchronous replication mean for the application developer?**

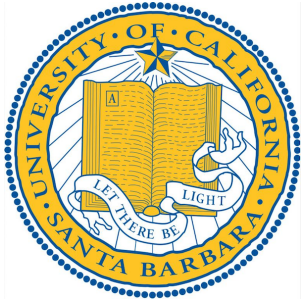




Reads vs. Writes

- Replication is asynchronous
  - This means writes will eventually appear on the slave, but not immediately.
  - Without this, it would be difficult/slow to scale.
- What does asynchronous replication mean for the application developer?
  - **In the context of a transaction, reads must still go to the master.**

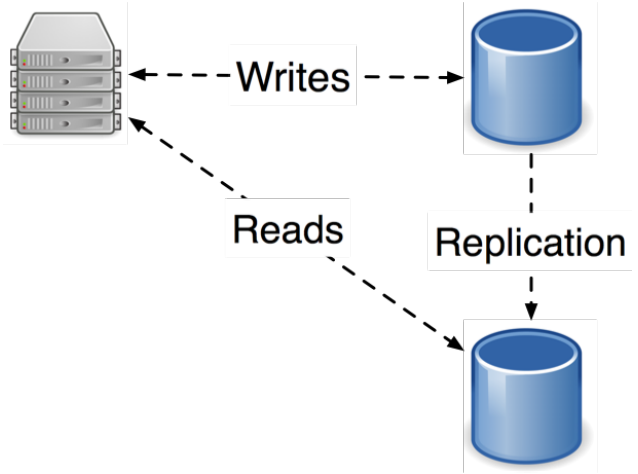


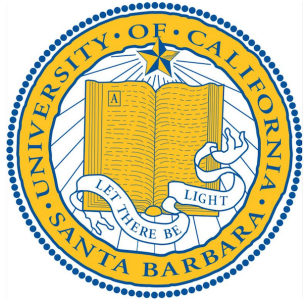


Reads vs. Writes

Two types of replication:

- Statement-level:
  - Similar to streaming the journal from the master to the slave
- Block-level:
  - Instead of sending the SQL statements to the slave, send the consequences of those statements
- Advantages of each?

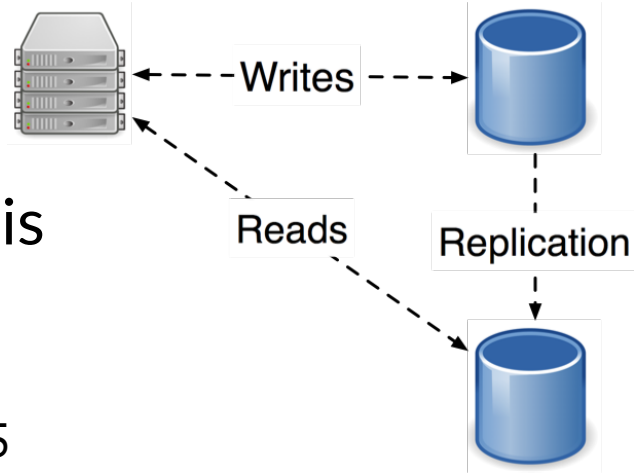


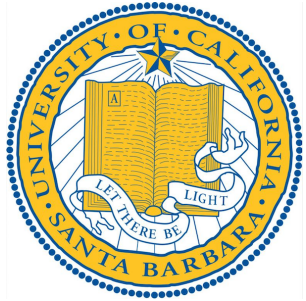


Reads vs. Writes

Advantages of statement-level replication:

- You send less data. A SQL statement is generally more compact than its consequences.
  - Example: `UPDATE txns SET amount = 5`
- SQL statements must now be deterministic
  - Example:
    - ```
      UPDATE txns
      SET amount = 5, updated_at = NOW()
      ```



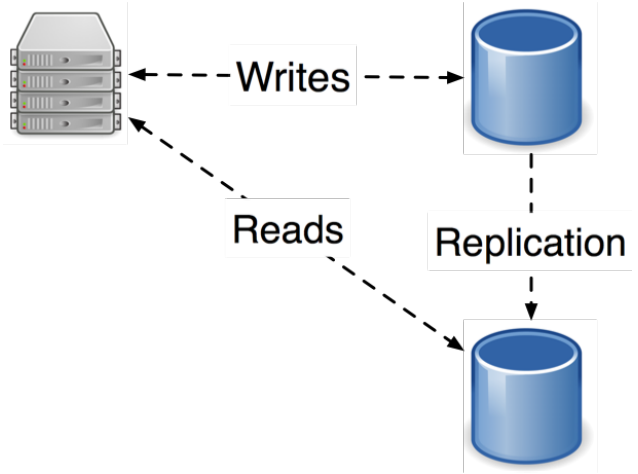

      ```

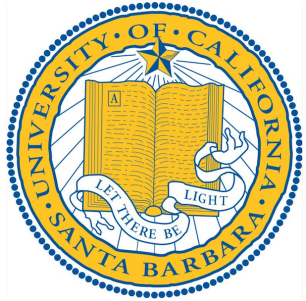
Reads vs. Writes

No built in support in Rails, but the Octopus gem can do this for you:

```
Octopus.using(:read_slave) do
  num_users = User.count
end

Octopus.using(:master) do
  User.create(:name => "Mike")
end
```

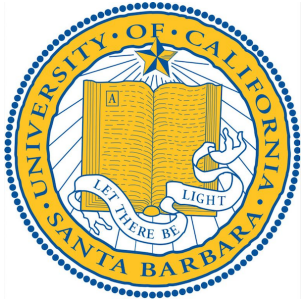




Reads vs. Writes

- Strengths:
  - For applications with a high read-to-write ratio, you can reduce the load on your master database significantly.

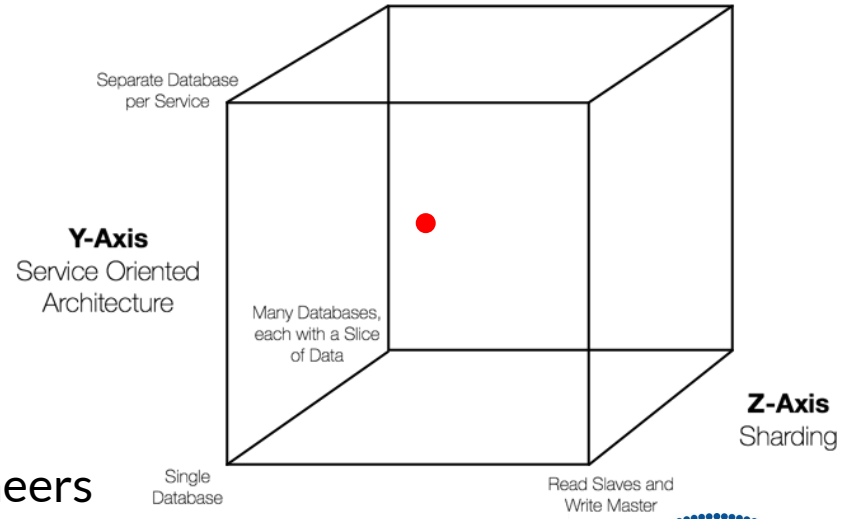
- Weaknesses:
  - Application developer needs to think about reads that affect writes vs. reads that don't affect writes



AKF Cube at Appfolio

At Appfolio...

- High usage of sharding
  - Each customer in separate database.
- Medium use of SOA
  - Some functionality broken out into services, but more for scaling engineers than scaling load.
- Low use of Read vs. Write distinction
  - We have read slaves for backup and analysis.

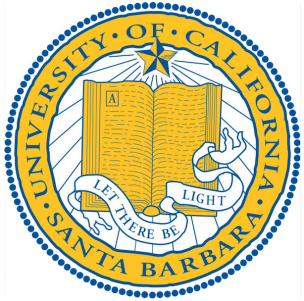


Conclusion

Horizontal scaling of relational databases is hard.

There is no silver bullet, but by combining sharding, SOA and read slaves you can get very far.

For applications that need to scale writes beyond what RDBMS can offer, you need non-relational data stores.



For Next Time...

Josep Blanquer, Chief Architect at RightScale:

"Beyond SQL: Non-Relational Data Stores"

Read "Eventually Consistent" and "CAP 12 Years Later" before Tuesday

Reminder: trip to Citrix Data Center next Thursday (Nov. 6th). Directions will be posted on website.