Django Under **Massive** Loads

Christophe Pettus (@xof)
PostgreSQL Experts, Inc.

thebuild.com
pgexperts.com

What is this talk?

- PostgreSQL Experts, Inc. is a database consultancy.
  - You probably guessed that.
- We also have an applications development practice.
  - We mostly do Django development.

PGX
POSTGRESQL
EXPERTS, INC.

Tales from the battlefield.

- We have clients who have very, very large Django sites.
- We've collected a lot of wisdom on how they managed to keep their sites up.
- This talk is a distillation of their wisdom.
- Others (especially us) have made all these mistakes, so you don't have to.

PGX
POSTGRESQL
EXPERTS, INC.

Selection bias noted.

- We're a PostgreSQL shop.
  - You probably guessed that, too.
- Thus, this will have some PostgreSQL-specific information.
- Most of it applies to other database products, too.

PGX
POSTGRESQL
EXPERTS, INC.

Audience.

- Application developers just getting started on the next big thing.
- … or whose site threatens to become the next big thing.
- … or who just don't want to appear in Hacker News in a bad way.

Django can do it.

- Django runs some of the busiest sites on the web.
  - Disqus, Instagram.
- If it can run those sites, it can run yours.
- Small optimizations can make a huge difference.

Structure.

- Tips and tricks.
- Mostly things not to do.
- Please ask questions!
- Please disagree!
- And now, let's start with…

The (Very) Front End

PGX
POSTGRESQL
EXPERTS, INC.

Front-end servers.

- Everyone obsesses about them.
- They don't matter.
- No, really, they don't matter.
- Once you've fixed everything else, worry about that.
- You've never fixed everything else.

OK, OK, fine.

- ngnix.
- gunicorn.
- + gevent.
- You now have a slide you can show your boss.
  - It's from an expert!

PGX
POSTGRESQL
EXPERTS, INC.

Party like it's 1999.

- Most of the time processing a request is after the first byte is received by the client.
- Keeping web pages small, clean and light will make more difference than almost anything else.
- Use HTML Boilerplate, Twitter Bootstrap? Trim, trim, trim to what you need.

Avoid “site pestering.”

- Avoid a large flurry of JavaScript requests back to the server from the initial page.
- Each one has the full round-trip latency of the first request.
- Reduce the amount of data you need to get, and batch the calls together.

Use a CDN for static content.

- Serving common static content is a terrible use of your bandwidth.
- CDNs can significantly improve your overall page-load time.
- Don't use for dynamic content: propagation rates are just too slow.
- If you can afford it, a caching CDN?

Use a front-end cache.

- ngnix, Varnish — or both!
- Don't use JavaScript callbacks on cached pages.
  - Rather defeats the purpose of a cache.
- Use JavaScript and HTML5 local storage for trivial customizations.
- Cookies defeat caching!

DNS Servers.

- A surprisingly large contributor to page-load time.
- Use a specialist DNS service.
  - EasyDNS is fast and cheap.
- Especially important if you have multiple subdomains on a single page.

The View Layer

PGX
POSTGRESQL
EXPERTS, INC.

Template-first design.

- Let the template drive your data acquisition.
- Don’t do ORM operations unless the particular template expansion actually needs it.
- Put QuerySets and callables, rather than evaluated data, in the template contexts.

Cache everything.

- Django has extensive template caching facilities. Use them.
  - Cache full pages if you can.
  - Cache fragments if you can't.
- Always use a memory-based cache.
  - memcached, Redis.

Cache results.

- QuerySets are serializable!
- Store them in an in-memory store.
  - Redis is great for basic queues, etc.
  - memcached if you only need a flat store.
- Remember thundering herds, etc.
  - Always opt to return stale data.

Consider full prerendering.

- Build entire page and cache on disk.
- Let the web server serve it directly.
  - Standard ngnix config will do this for you with appropriate path settings.
- Or let ngnix or Varnish do the caching.

Returning large files.

- Use X-Accel-Redirect or equivalent.
- Never hand the large file directly back through Django.
  - Never. Write it to disk if you have to.
- Especially important if using back-end worker servers like gunicorn.

Middleware.

- Keep the middleware stack under control.
- Do you really need this to run on every request?
- Don’t use TransactionMiddleware…
  - Control transactions using decorators.
  - https://github.com/Xof/xact

Defer everything.

- Do not run asynchronous tasks in your view functions.
  - Send mail, fetch other sites, etc.
  - Queue those for later processing.
  - Queue synchronous tasks if they are long-running.
    - Generate a “best-guess” result first.

The Model Layer

PGX
POSTGRESQL
EXPERTS, INC.

Model-building.

- Keep models simple and focused.
- Use natural keys (instead of AutoField) whenever possible.
- Don’t be afraid of foreign keys.
- Do not have frequently-updated singleton rows.

Fast vs slow data.

- A single logical object can have both "fast" and "slow" sections:
  - Username vs last access time.
- Separate these into different tables.
- Avoids a large class of foreign key locking issues.

Result prefetching.

- QuerySets will fetch the *entire* database result set the first time they need a *single* row.
  - … at least using psycopg2.
- Make sure database result sets are small.
- Do not rely on QuerySet slicing.

QuerySet caching.

- QuerySets retain their iterated-over results until released.
- This can be a significant memory sink.
- Release QuerySets once you are done with them.
  - But if can you store the results for future use? Do it.

Transactions.

- Django's default transaction handling isn't good for high-load sites.
  - Or interdependent models.
- TransactionMiddleware is better, but adds gratuitous transactions to read-only operations.
  - That messages up pgPool II, etc.

Better transactions.

- Control transactions precisely around blocks of code that need it.
- autocommit = True
- Standard Django decorators
  - https://github.com/Xof/xact

PGX
POSTGRESQL
EXPERTS, INC.

Using transactions.

- Keep transactions short and to the point.
- Like any good writing, start as late as you can, finish as early as you can.
- Never rely on Django to clean up transactions.
- Never wait for an asynchronous event with an open transaction.

More friendly advice.

- Do not iterate over large QuerySets…
  - … especially while doing updates back to the database.
- Do joins in the database, not in Python.
  - Don't be afraid of writing custom SQL if that's what it takes.

The Database

PGX
POSTGRESQL
EXPERTS, INC.

Databases are your friend.

- Keep queries short and stylized.
- Don’t let your users assemble arbitrary queries.
  - It’s almost impossible to optimize for this case.
- Denormalization is not (always) evil.

Do not do this.

- Store sessions in the database.
- Store your task queue in the database.
  - Especially if your task queue runner polls the database.
  - (I'm looking at you, Celery.)
- Store high-volume data in an otherwise-transactional database (clickstream, etc.)

Never lock.

1. Never issue an explicit database lock.
2. If you think you have to do it, see above.
3. If you are absolutely sure you have to do it, see point number 1.

- Explicit locking is a near-guarantee of an application architecture problem.

Connection poolers.

- Django has no built-in connection pooling.
- Connection time can be larger than the request processing time.
- pgbouncer vs pgPool II.
- https://github.com/gmcguire/django-db-pool
- Either session or transaction mode.

Database load balancing.

- If using PostgreSQL, use 9.x's streaming replication.
- Ideally designed for web-type read vs write loads.
- How to route requests to the right servers?

Django database routing.

- Use Django database routing to distribute writes to the master, reads to the secondaries.
- If more than one secondary, use pgPool II or a TCP/IP-based load balancer (HAProxy).
- Remember replication lag issues.

pgbouncer

- Developed by Skype.
- Very fast and light-weight.
- Good for a single database or a self-managed replication set.
- No failover, load balancing, or automatic query routing.

pgPool II

- More sophisticated, slower than pgbouncer.
- Does failover and query routing for a replication set.
- Does not interact well with standard Django transaction management.
  - Use xact.

System Components

PGX
POSTGRESQL
EXPERTS, INC.

Full-text search.

- Solr and Elastic Search are very powerful.
- They can be significant time-sinks.
- Consider using the DB's full-text search.
  - PostgreSQL's is very powerful.
  - MySQL's exists.
- *Always* precalculate and cache results.

In-memory databases

- memcached is a great flat in/out store.
- Redis is great for queuing and more advanced operations.
- Be sure to adjust timeouts in case of server failure.

Welcome to the cloud.

- Just about everyone develops or launches on a cloud provider now.
- Cloud services range from super-managed (Heroku) to here's-your-machine (Linode).
- Carefully compare costs with what you get.
  - You *will* need operations staff for a large site, no matter where you host.

Cloud good.

- Cloud provider keeps the lights on.
- Higher-touch providers configure and manage your system.
- You can spin up and spin down virtual hosts to manage front-end server demand.
- Database demand? Not so much, even with fast-sharding solutions.

Cloud bad.

- Highly unpredictable I/O performance.
  - It varies from pretty bad to really, really horrible.
  - Get lots of RAM.
- Cloud providers vary considerably in how seriously they take your particular virtual machine.

Push the button.

- Always automate deployments and provisioning.
- Puppet, Chef, Fabric…
  - Pick one, learn it well, use it everywhere.
- The multiple components involved in high-performance sites are a nightmare to administer by hand.

Monitor, monitor, monitor.

- Monitor everything.
  - CPU usage.
  - Memory usage.
  - Disk space.
  - Replication lag.
- New Relic if you can afford it.

Sharding

PGX
POSTGRESQL
EXPERTS, INC.

Sharding.

- Sharding, defined:
  - Dividing the database layer over multiple servers, none of which have the entire set of data.
- Huge sites eventually need to shard.
- Sharding always affects application architecture.

PGX
POSTGRESQL
EXPERTS, INC.

Sharding strategies.

- Data-driven sharding.
  - By customer, by geography, etc.
- "Anonymous" sharding.
  - Customers whose ID ends in 9 go on this server.
- Common data distributed across all servers.

Challenges.

- Route queries to the right server.
- Doing aggregation across all servers.
- Distributing common data to all servers.
- Bringing up new shards.

Designing for sharding.

- Understand the growth model of each part of your data.
  - Linear with user base.
  - Exponential with user base.
  - Linear over time.
  - Constant.

Strategies.

- Linear with users: Isolate into specific Django applications for later migration to shards.
- Exponential with users: Eliminate if possible.
- Constant and linear with time: Segregate as future common data.

Other sharding fun.

- You will need to encode shard identity into keys.
  - See Instagram's blog for one strategy.
- Shard migration: Tied to distribution strategy.
- Pushing common data to all servers.
  - Dealing with lag.

Consolidated data.

- Sharding requries data consolidation for reporting, etc.
- Strategies:
  - A single consolidation that queries the shards (pull model).
  - Shards that push their data to an aggreation server (push model).

To avoid.

- A single-point-of-failure "master" server.
  - Just moves the load problem around.
- Inter-shard communication.
  - Doing queries and joins across shards.
  - Each shard should be a "mini-you."

Design your application for sharding.

- It will make the overall application cleaner.
- You'll learn a great deal more about your data model.
- When it comes time to shard, you will be a hero.

PGX
POSTGRESQL
EXPERTS, INC.

Summary!

PGX
POSTGRESQL
EXPERTS, INC.

I thought he'd never stop.

- Django can handle massive, server-melting loads.
- There's no one trick; it's a collection of small things and avoiding pitfalls.
- Focus on keeping your app lean.
  - You can hardware your way out of (almost) all the rest.

Thank you!

Questions?

thebuild.com

pgexperts.com

@xof

PGX
POSTGRESQL
EXPERTS, INC.