How To Scale Django: Finding the Bottleneck

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=253

Django is an excellent Python based platform for building modern web apps. One of its biggest strengths is that it helps developers work faster.

You've built your awesome app and deployed it. Things are great, but now that you're loading it up with larger amounts of data and you're starting to have several people use it at the same time, it's not as fast as you'd like.

It's a common problem. Fortunately, we have some tools to help alleviate the problems.

First, let's check for a few of the more obvious issues:

Use a Real Database

During local development, it's hard to beat SQLite3. Unless you're careful, you might be using it on your virtual server as well.

SQLite3 doesn't scale to multiple simultaneous users like MySQL and PostgreSQL do, especially for operations that do many write operations (if you're using sessions then you are writing to the database).

If you're using a low-memory, for example the 512MB, I'd suggest using MySQL. If you have memory to spare (2GB or more), then I'd urge you to consider PostgreSQL, since it is preferred by many of the Django developers.

Disable Debug Mode

Debug mode is absolutely necessary when doing local development, but it will slow down your production server. Look at your virtual server's settings.py and check to ensure that DEBUG is set to False. Also confirm that TEMPLATE_DEBUG is also set to False or that it is set to DEBUG.

Use the Debug Toolbar to Pin Down Performance Problems

On your local development computer, not your production server, turn on [Django Debug Toolbar]() to locate specific problems.

You do this by installing the **django-debug-toolbar** module and then adding an entry to your MIDDLEWARE_CLASSES dictionary like this:

```
MIDDLEWARE_CLASSES = (
    # ...
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    # ...
)
```

You'll also need to create an INTERNAL_IPS variable and add your IP address. If you're developing locally, your IP address is probably 127.0.0.1, so you'd add a line like this to your settings.py:

```
INTERNAL_IPS = ('127.0.0.1',)
```

Finally, add **debug_toolbar** as the last item in your INSTALLED_APPS, like this:

```
INSTALLED_APPS = (
    # ...
    'debug_toolbar',
)
```

The installation documentation contains some more detail and optional configuration options you may want to consider.

Remember, don't push these changes to production on accident! (If you do it on purpose, that's fine).

Now you should see a black panel appear down the side of your web pages as you browse around the site. If you like stats and numbers and all kinds of geeky details, you'll love it. You'll also quickly see why you don't want this on your production server!

Having built numerous Django apps, I'll suggest you narrow in on the SQL section of the panel, since that is commonly an area of concern.

One benefit/problem that Django has is lazy loading of related fields when queries are performed. This means that Django will prefer not to do a join. If you end up needing related fields, it will do an additional query each time a related field is needed.

If you do need related fields, this can causes n+1 number of SQL queries. For example, let's say you are going to make a replacement version of Twitter. You create a model for tweets and each model is related to the User model. On your homepage, you list the 30 latest tweets along with the name of the user who created them. This can cause you to do at least 31 SQL queries. One query to get the list of tweets, and one query for each username lookup.

The solution to this problem is select_related. This is a very simple modification to the query that causes Django to do a join when fetching data. You should use it on any lookup where you know you'll need related fields.

You simply modify a query like this:

```
Entry.objects.get(id=5)
```

to look like this:

```
Entry.objects.select_related().get(id=5)
```

Read the documentation for this feature and use it only when necessary.

Wrap Up

In my experience, the above issues solve many sites' initial performance problems and are all quite easy to fix with minimal code and configuration changes.