django in the real world

yes! it scales!... YAY!

Israel Fermin Montilla

Software Engineer @ dubizzle

December 16, 2017

from iferminm import more_data

- Software Engineer @ dubizzle
- Venezuelan living in Dubai, UAE
- T: @iferminm
- blog: http://iffm.me

What will we see in this talk?

- Pareto Principle
- The simple django project
- Measuring
- Common bottlenecks

Basic concepts: Pareto principle

Basic concepts: Pareto principle

The Pareto principle states that, for many events, roughly 80% of the effects come from 20% of the causes

–Wikipedia

Basic concepts: Pareto principle

The Pareto principle states that, for many events, roughly 80% of the effects come from 20% of the causes

–Wikipedia

For example: 20% of the code produces 80% of the bugs.

Initial django project in production

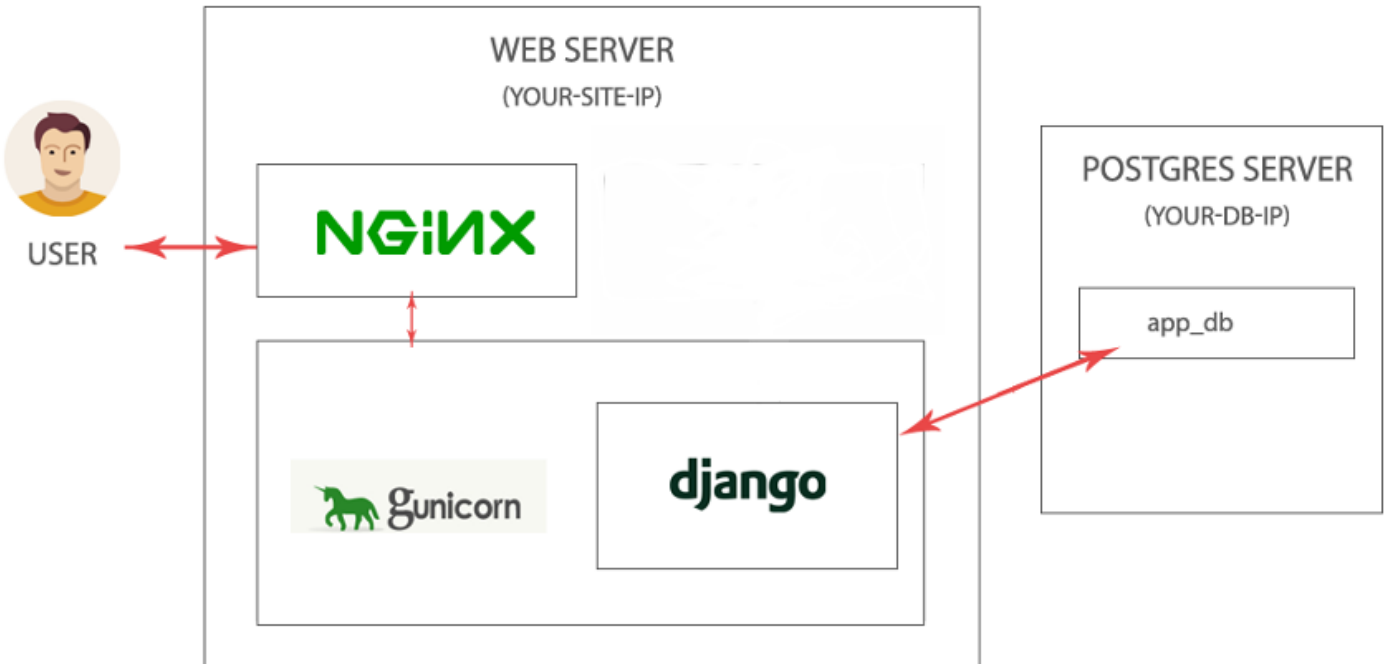

Figure: Basic django project production setup

Profile first

django-debug-toolbar

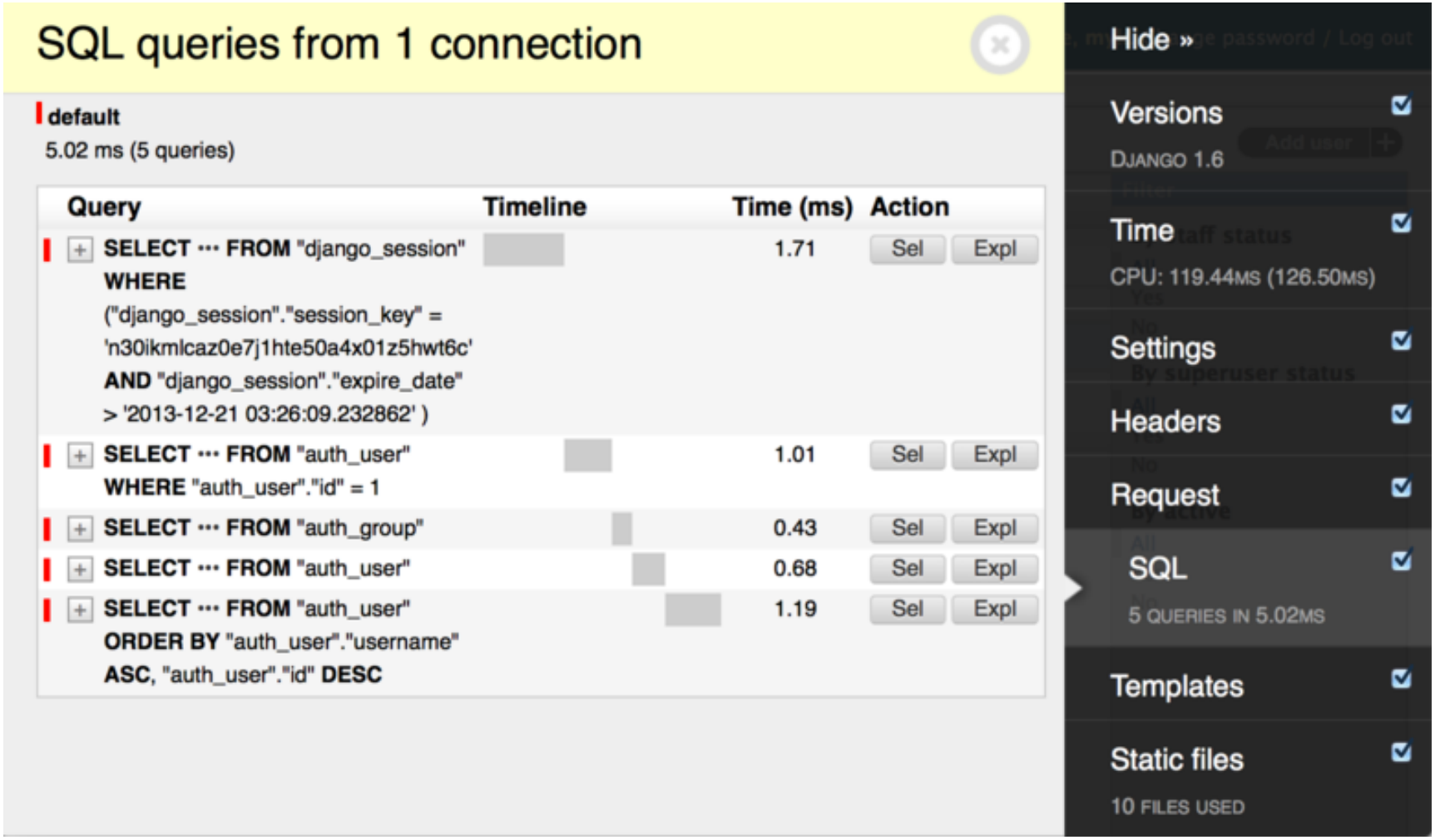

Figure: debug_toolbar in action

cProfile + snakeviz

Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000421	0.000421	0.000421	0.000421	~:0(<built-in method listdir>)
1	0.000104	0.000104	0.000202	0.000202	functools.py:441(wrapper)
1	7.9e-05	7.9e-05	0.000294	0.000294	fnmatch.py:48(filter)
1	6.7e-05	6.7e-05	8e-05	8e-05	functools.py:342(_make_key)
1	4.4e-05	4.4e-05	0.00079	0.00079	glob.py:61(glob1)

Figure: snakeviz list view

cProfile + snakeviz

Name:
`filter`
Cumulative Time:
`0.000294 s (31.78 %)`
File:
`fnmatch.py`
Line:
`48`
Directory:
`/Users/jiffyclub/miniconda3/envs/snakevizdev/lib/python3.4/`

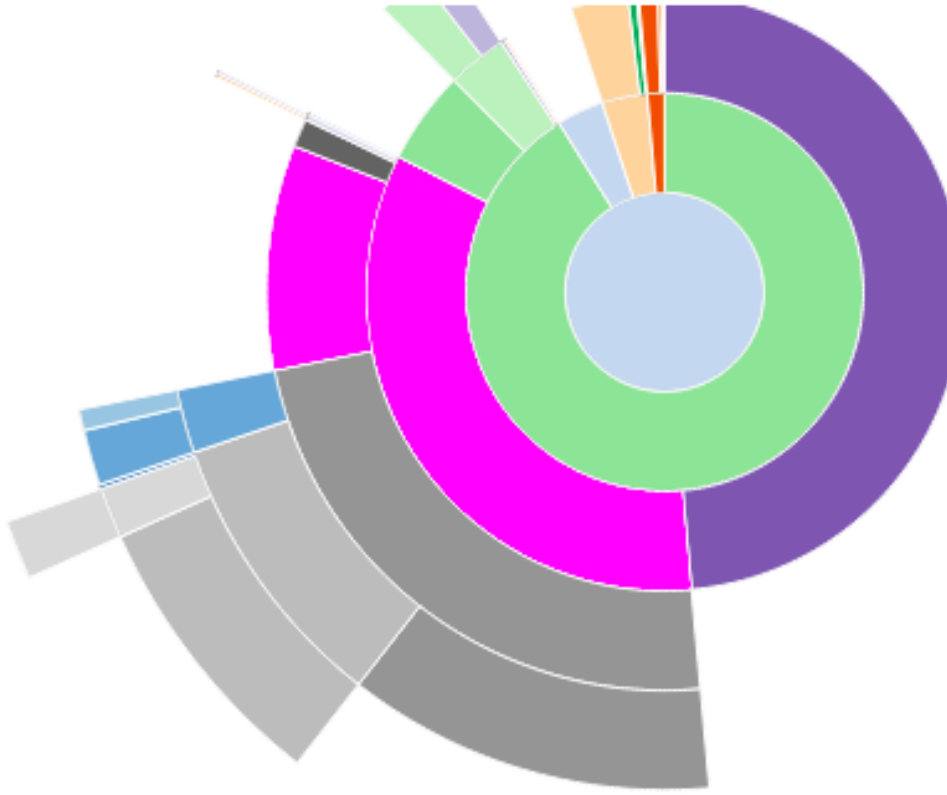

Figure: snakeviz sunburst diagram

vprof

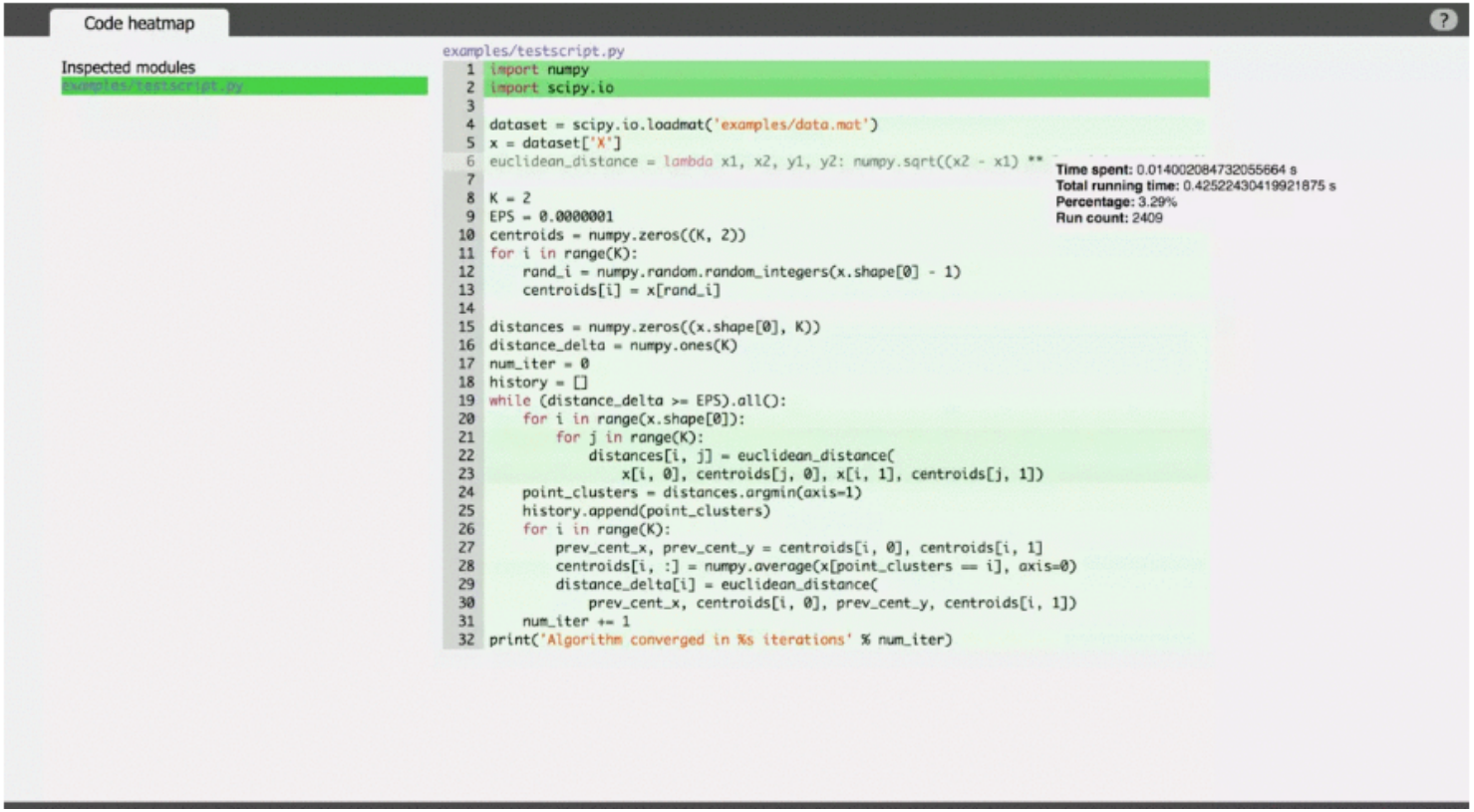

Figure: vprof code heatmap

vprof

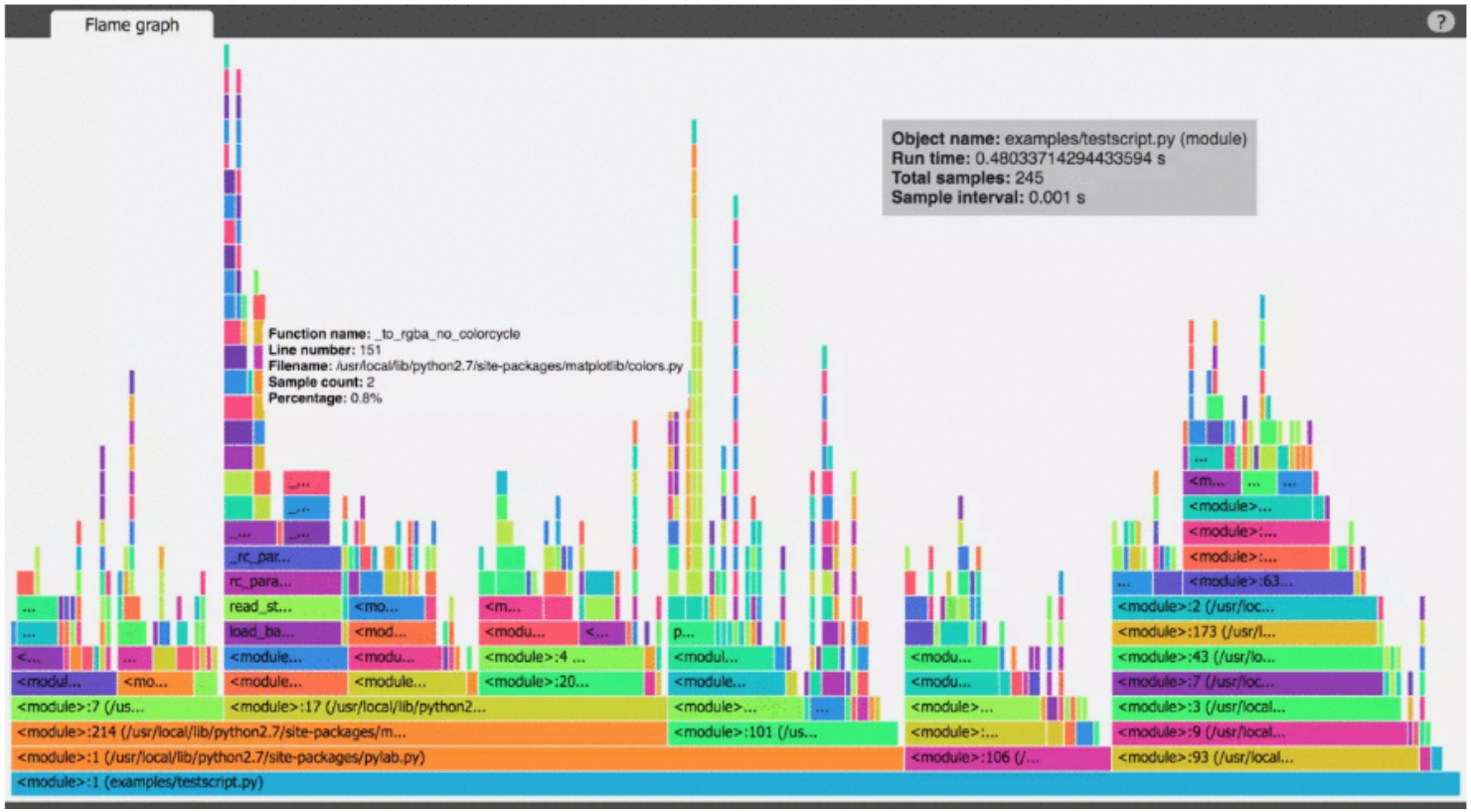

Figure: vprof flame diagram

vprof

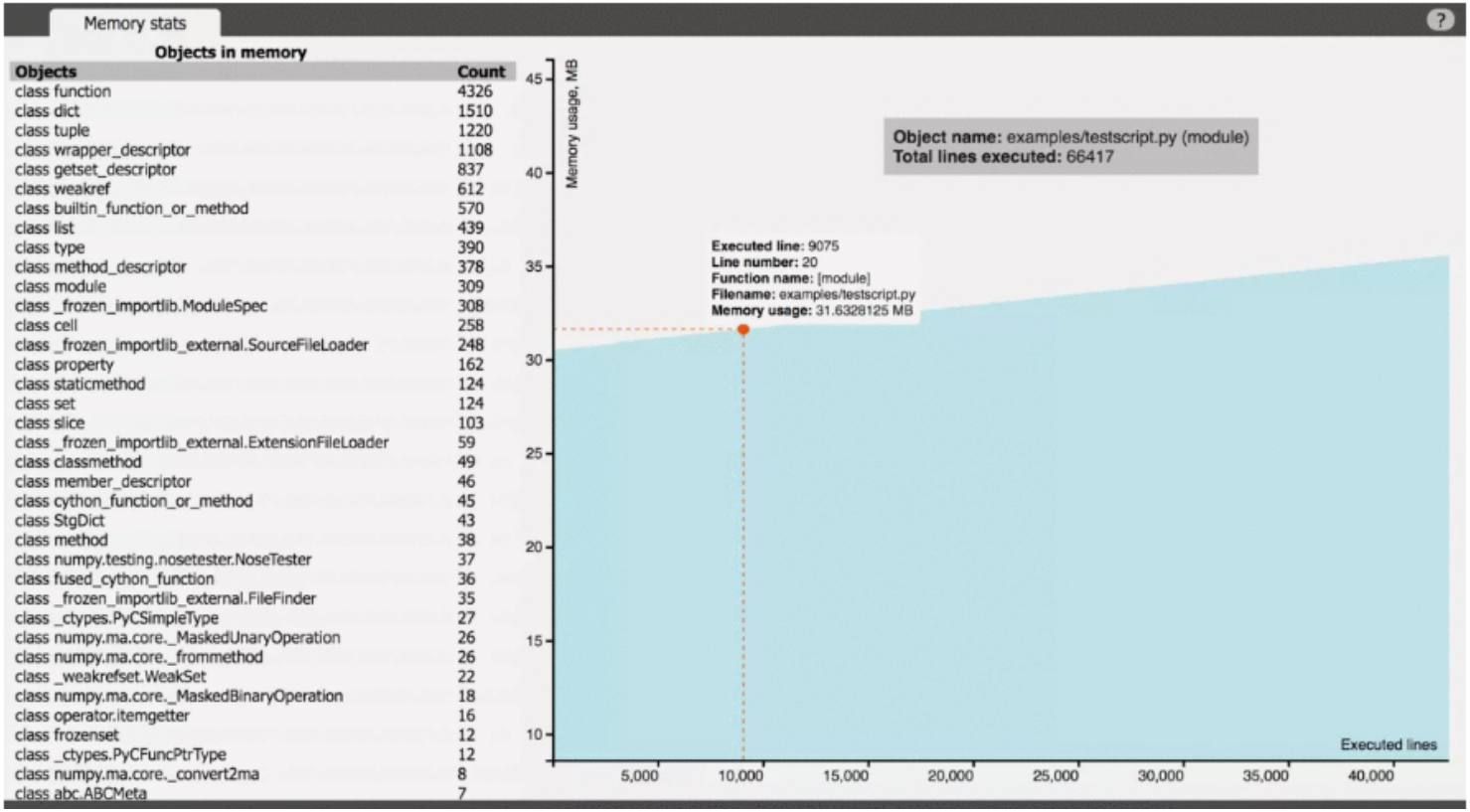

Figure: vprof memory profiler

vprof

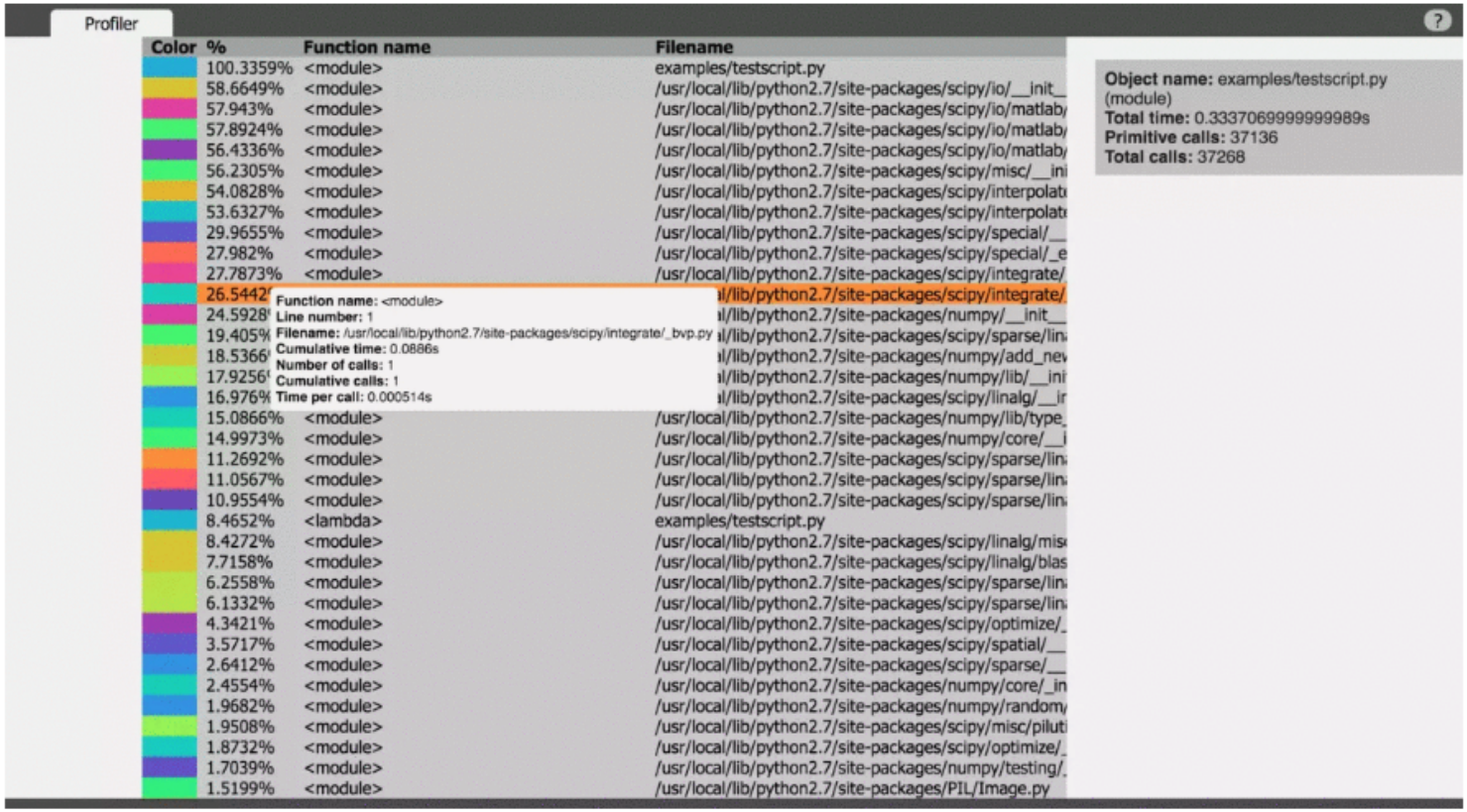

Figure: vprof profiler

newrelic

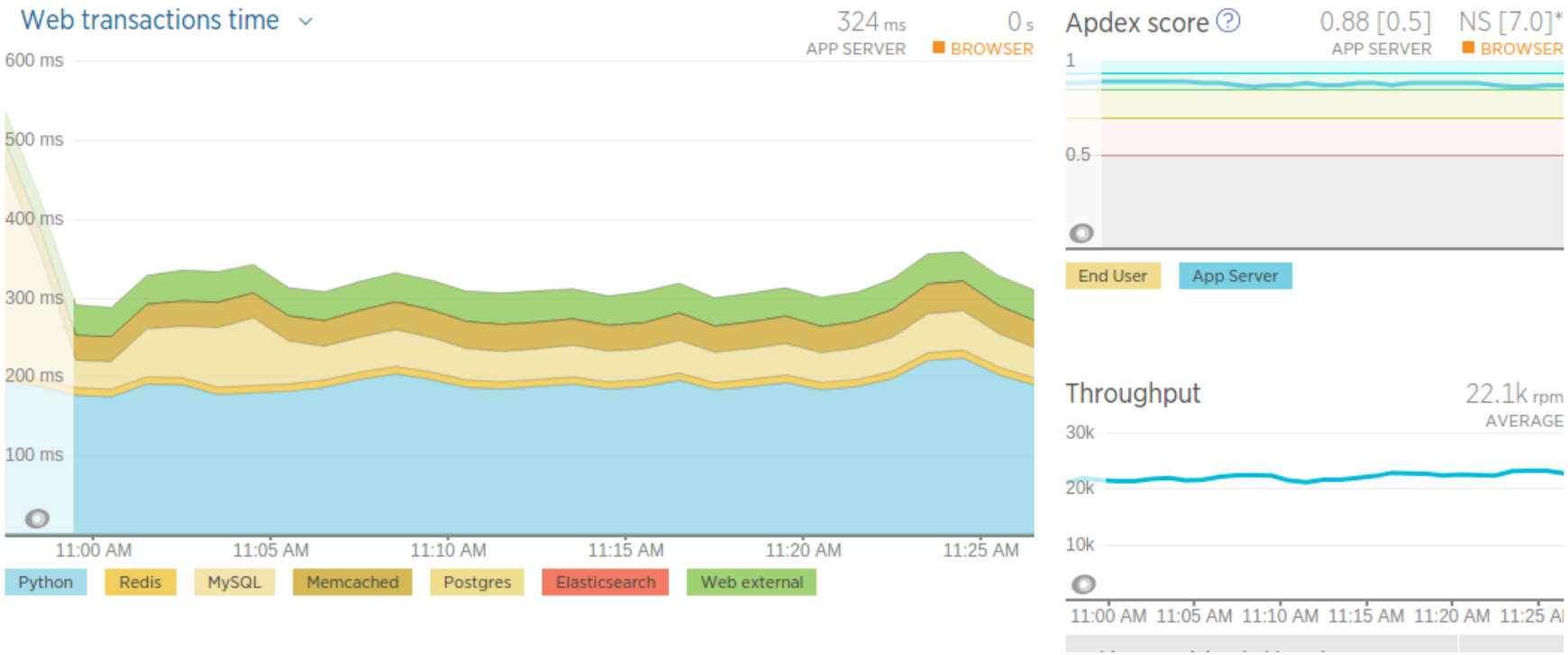

Figure: Part of newrelic's main dashboard

newrelic

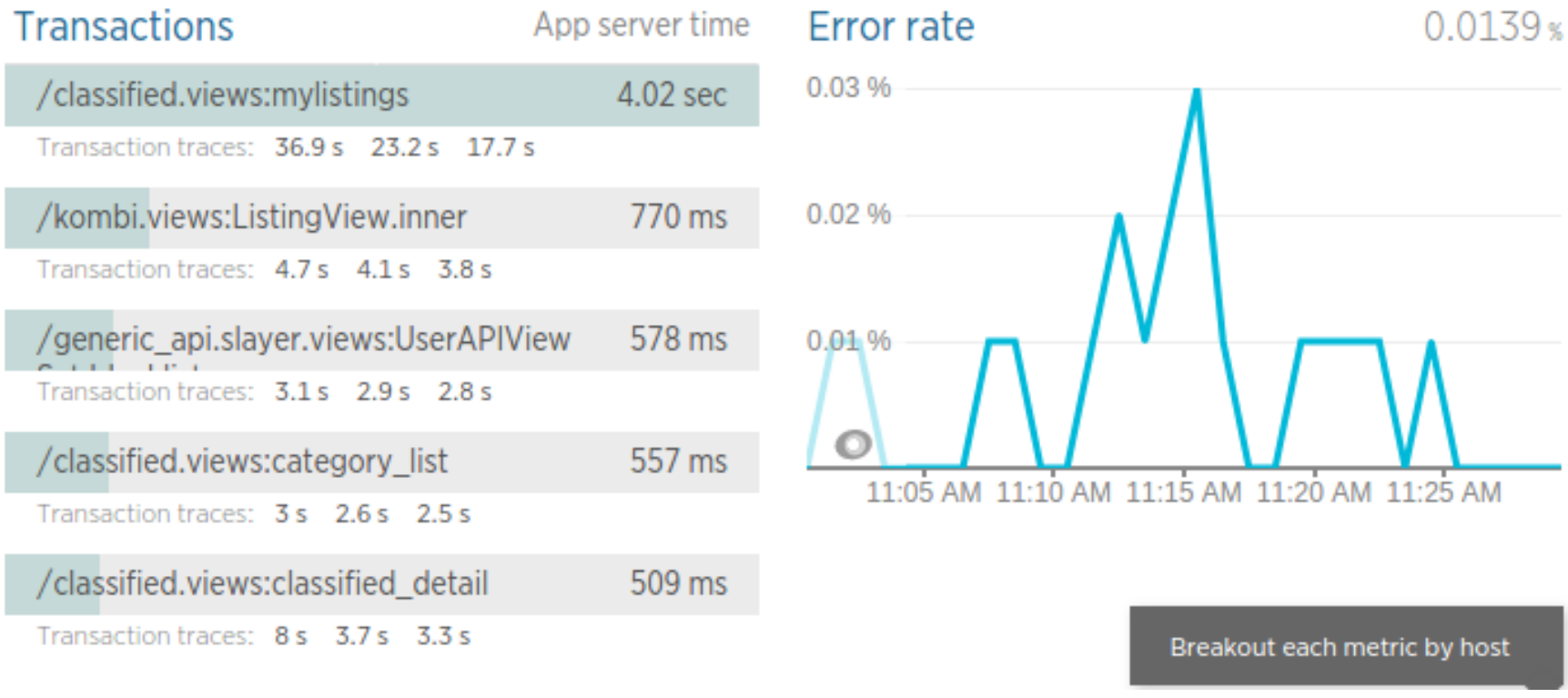

Figure: part of newrelic's main dashboard

newrelic

Breakdown table

Category	Segment	% Time	Avg calls (per txn)	Avg time (ms)
Function	kombi.views:ListingView.inner	63.1	1.0	491
Database	Memcached get	4.3	36.7	33.4
Database	Memcached set	1.1	4.76	8.87
Function	MySQLdb:Connect	0.9	1.98	6.86
Database	MySQL classified_classified_au select	0.8	2.09	6.06
Database	MySQL classified_propertyforrent_rs select	0.7	1.91	5.62
Database	MySQL classified_classified_fu select	0.3	1.95	2.26
Database	MySQL classified_propertyforsale_rs select	0.3	1.86	2.54

Show all segments →

Figure: Inside a web transaction in newrelic

Database

Reduce query counts

```
subs = Subscription.objects.filter(
    user_id=user.pk
)
for s in subs:
    packages.append(s.package.name)
```

Reduce query counts

```
subs = Subscription.objects.filter(
    user_id=user.pk
)
for s in subs:
    packages.append(s.package.name)
```

N hits to the database

Reduce query counts

```
subs = Subscription.objects.filter(
    user_id=user.pk
)
for s in subs:
    packages.append(s.package.name)
```

N hits to the database

```
subs = Subscription.objects.filter(
    user_id=user.pk
).select_related('package')
```

Reduce query counts

```
subs = Subscription.objects.filter(
    user_id=user.pk
)
for s in subs:
    packages.append(s.package.name)
```

N hits to the database

```
subs = Subscription.objects.filter(
    user_id=user.pk
).select_related('package')
```

Will join the table and return it in one hit

Reduce query counts

- select_related
- prefetch_related

Reduce query counts

- Use it wisely and measure

```
user = User.objects.select_related(
    'sodas'
).get(pk=request.data['user_id'])

# No additional query
user.sodas.all()
```

Reduce query counts

- Use it wisely and measure

```
user = User.objects.select_related(
    'sodas'
).get(pk=request.data['user_id'])

# No additional query
user.sodas.all()
```

```
# Triggers an additional query
user.sodas.filter(name='pepsi')

# Sometimes it's better to use the cached
    result
# and filter in memory
[s for s in user.sodas.all() if s.name ==
    'pepsi']
```

Reduce query counts

Use the Prefetch object!

```
# A product has many subscriptions and
# a subscription can have many products

queryset = Subscription.objects.filter(
    status=expired
).select_related('credits')

prefetch = Prefetch('subscriptions',
                    queryset=queryset)
products = Product.objects.prefetch_related(
    prefetch
).filter(section='jobs')
```

Reduce query time

Reduce query time

- Indexing

Reduce query time

- Indexing

```
class UserProfile(models.Model):
    user = models.ForeignKey('auth_user')
    dob = models.DateField(db_index=True)
    external_id = models.IntegerField(
        db_index=True
    )
```

Reduce query time

- Indexing

```
class UserProfile(models.Model):
    user = models.ForeignKey('auth_user')
    dob = models.DateField(db_index=True)
    external_id = models.IntegerField(
        db_index=True
    )
```

Note: Your DBMS updates your indices in *write time (INSERT and UPDATE)*

Some notes on indexing

- You need to measure before you do it. Run EXPLAIN on the query (Seq scan)
- Index by workload
- If you filter on multiple columns use index_together Meta option
- Check if the index is used before you push it. Run EXPLAIN again

Expensive JOINs

Sometimes you might want to separate them into two different queries.

```
# You may want to see the credit spending
    behavior of your users
Credit.objects.filter(
    subscription__pkg__type='motors'
).select_related('resource')

# Sometimes two queries might perform better
subs_ids = Subscription.objects.filter(
    pkg__type='motors'
).values_list('id', flat=True)

Credit.objects.filter(
    subscription_id__in=subs_ids
).select_related('resource')
```

ALWAYS MEASURE

Avoid whole table COUNT() queries

After some point, having exact numbers is not important

```
1 PropertyForRent.objects.count()
```

Avoid whole table COUNT() queries

After some point, having exact numbers is not important

```
PropertyForRent.objects.count()
```

You can instead do a raw SQL query

```
# Postgres
SELECT reltuples FROM pg_class
    WHERE relname = 'property_for_rent'

# MySQL
SELECT table_rows FROM information_schema.
    tables
    WHERE table_schema = DATABASE()
    AND table_name = 'property_for_rent'
```

Avoid whole table COUNT() queries

After some point, having exact numbers is not important

```
PropertyForRent.objects.count()
```

You can instead do a raw SQL query

```
# Postgres
SELECT reltuples FROM pg_class
    WHERE relname = 'property_for_rent'

# MySQL
SELECT table_rows FROM information_schema.
    tables
    WHERE table_schema = DATABASE()
    AND table_name = 'property_for_rent'
```

This could reduce up to 90% response time

Use persistent connections

```
DATABASES = {
    'default': {
        # The usual...
        'CONN_MAX_AGE': None,
    }
}
```

Know your ORM

- Read the full ORM docs at least once
- Use F expressions to reference values within the queryset
- Use Q expressions for advanced filters
- Explore the aggregation framework
- Use values(), values_list(), only() and defer() when the results are too big

Denormalize

- Evaluate huge joins
- Don't use Generic Relations

Denormalize

- Evaluate huge joins
- Don't use Generic Relations

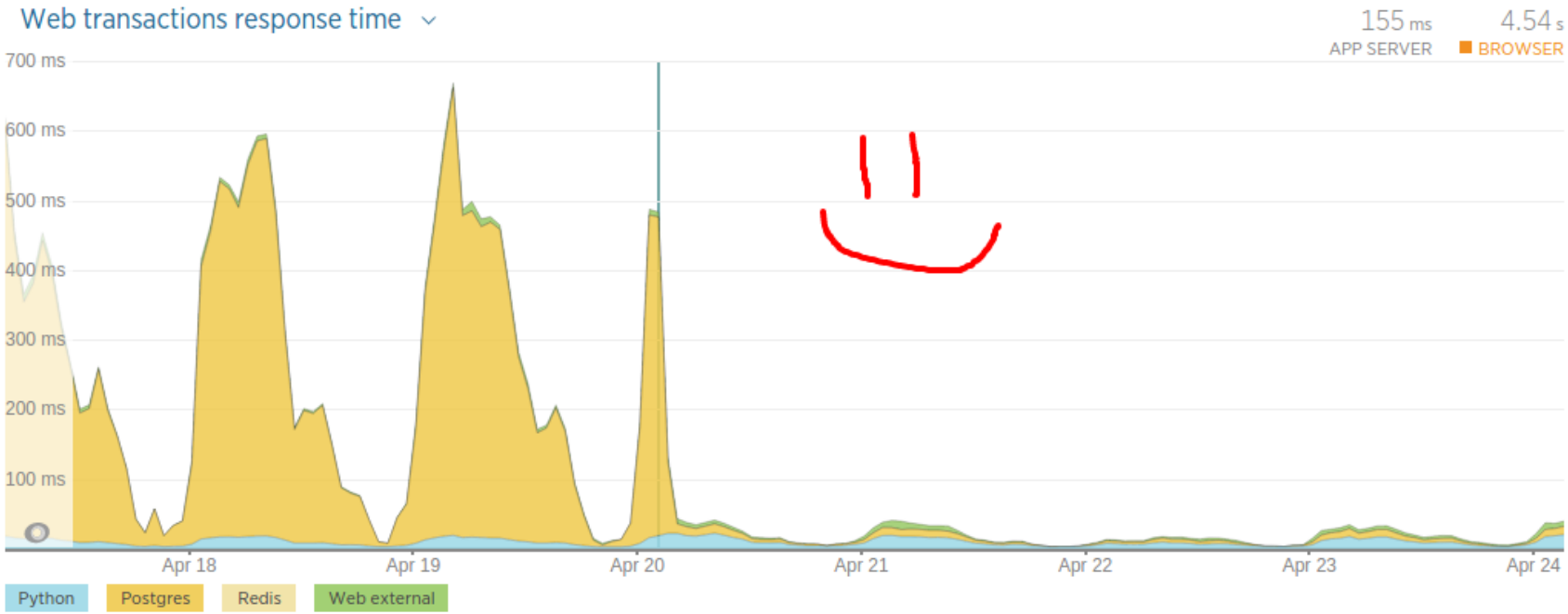

Figure: Response time reduction after denormalizing a Generic Relation

Query caching

- johny-cache
- django-cache-machine

Templates

Russian Doll Caching

Russian Doll Caching

```
{% cache MIDDLE_TTL "ads" request.GET.page %}
    {% include "sections/property/postheader.html" %}
    <div class="ads-list">
    {% for ad in ads %}
        {% cache LONG_TTL "ad_description" ad_id ad.last_updated %}
            {% include "sections/property/ad_teaser.html" %}
        {% endcache %}
    {% endfor %}
{% endcache %}
```

Further Optimization

Further optimization

- Minimize your CSS and JS (django-compressor, webassets or django-pipeline)
- Optimize your static images
- Optimize user uploaded images
- Serve your media and static content from a CDN
- Do slow work later... (celery or python-rq)
- Use slave replicas for *read operations (and database routers)*


DESPACITO

Thank you!

**** We're hiring ****
**** iferminm@gmail.com ****