Integrated cache invalidation for better hit rates

FOSDEM PGDay 2014
Brussels, Belgium

Magnus Hagander
magnus@hagander.net

Magnus Hagander

- PostgreSQL
  - Core Team member
  - Committer
  - PostgreSQL Europe
- Redpill Linpro
  - Infrastructure services
  - Principal database consultant

We've all heard

"

There are only two hard
things in Computer Science:

Cache invalidation and naming things.
"

Phil Karlton

Scenario

- We're building a website
  - (Sorry all you backend people!)
- Let's do a webshop!

Picking our poisons

- PostgreSQL
- Django
  - Whatever webserver you choose
- Varnish

Simple database schema

- We'll ignore most requirements
- No payments
  - But we'll have dynamic pricing
  - And inventory
- Just articles and groups

Simple database schema

```
class ArticleGroup(models.Model):
  groupname = models.TextField(null=False)

class Article(models.Model):
  articlename = models.TextField(null=False)
  price = models.DecimalField(decimal_places=2,
                              max_digits=10, null=False)
  stock = models.IntegerField(null=False)
  group = models.ForeignKey(ArticleGroup)
```

Simple database schema

```
CREATE TABLE myshop_articlegroup(
  id SERIAL NOT NULL PRIMARY KEY,
  groupname text NOT NULL);

CREATE TABLE myshop_article(
  id SERIAL NOT NULL PRIMARY KEY,
  articlename text NOT NULL,
  price numeric(10,2) NOT NULL,
  stock int NOT NULL,
  group_id int NOT NULL
    REFERENCES myshop_articlegroup(id));
```

Basic views

- We need to be able to view an article
- Any (web) designers in the audience?

Basic views

```
def index(request):
    articles = Article.objects.all()
    return render_to_response('index.html', {
        'articles': articles,
        })
```

Basic views

```
def article(request, articleid):
    article = get_object_or_404(Article, id=articleid)
    return render_to_response('article.html', {
        'article': article,
        })
```

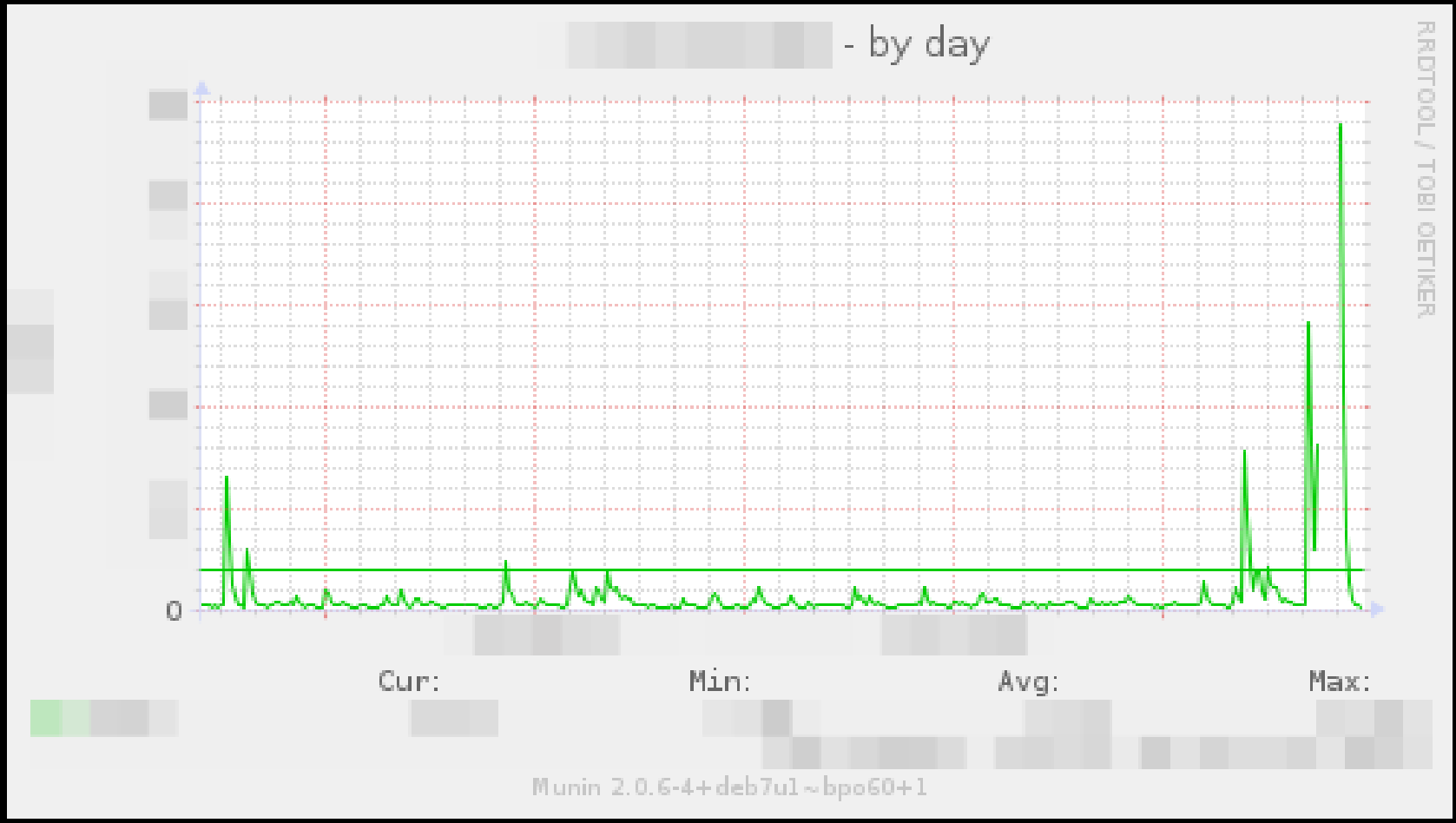
Basic views

```
urlpatterns = patterns('',
    url(r'^$', 'myshop.views.index'),
    url(r'^article/(\d+)/$', 'myshop.views.article'),
    ...
```

Works for a while

- Fully dynamic actually scales reasonably well
  - Modern servers are surprisingly fast
- We can tweak some simple things
  - Static assets to CDN, for example
    - Helps with bandwidth, but not much more

And then...


- by day
RRDTOOL / TOBI OETIKER
0
Cur:
Min:
Avg:
Max:
Munin 2.0.6-4+deb7u1~bpo60+1

Typical solutions

- More servers
- Even more servers
- Yet more servers
- (this can get expensive)

Typical solutions

- Cache!
- Faster deliveries and less load
- This is what we'll look at here

Caching

- Put a Varnish instance in front
  - (obviously >1 for redundancy, but..)
- Control cache time from application

Caching

```
@cache(minutes=30)
def article(request, articleid):
...

@cache(minutes=5)
def index(request):
...
```

Caching

```
def cache(hours=0, minutes=0, seconds=0):
    def _cache(fn):
        def __cache(request, *_args, **_kwargs):
            resp = fn(request, *_args, **_kwargs)
            td = datetime.timedelta(hours=hours,
                                    minutes=minutes,
                                    seconds=seconds)
            resp['Cache-Control'] = 's-maxage=%s' % td.seconds
            return resp
        return __cache
    return _cache
```

Caching

```
mha@mha-laptop:~$ curl -I http://localhost:9000/article/2/
```

```
HTTP/1.0 200 OK
Date: Sun, 19 Jan 2014 14:20:37 GMT
Server: WSGIServer/0.1 Python/2.7.3
Content-Type: text/html; charset=utf-8
Cache-Control: s-maxage=1800
```

Problems!

- Our inventory is now out of date
  - Up to 30 minutes!
- Our article list is now out of date
  - Up to 10 minutes!
- Our prices are out of date!
  - Everybody uses dynamic pricing, right?

Quick-fix

- Decrease cache-times
- Whatever is the maximum acceptable
  - 30 seconds? 10 seconds? 1 second?
- Leads to bad cache hitrates
- Still helps with peak-removal
- But only for popular articles

Better fix

- Forced cache invalidation
- Leave data in cache for a long time
- Only remove when it actually *changes*
  - But remove it quickly

Simple

- Trap object saving in Django
- Generate request to Varnish
- Varnish purges object

Simple

```
class Article(models.Model):
    ...
    def save(self, *args, **kwargs):
        super(Article, self).save(*args, **kwargs)
        conn = httplib.HTTPConnection('localhost:9001')
        conn.request('PURGE', '/article/%s/' % self.pk)
        conn.getresponse()
        conn.close()
```

Ouch!

- Did that make your eyes hurt?
  - Code in the model!
  - Model needs to know about URLs
  - Still didn't purge the index page
  - What if the change didn't come through Django

Code in the model

- This is fairly easy to fix
- Use signals or wrappers, etc

Model needs to know about URL

- Models should not care about this
- Maybe even *views* shouldn't
- Need to decouple and use something else
  - URLs are too dynamic

Still didn't purge the index page

- Same basic problem
- Model needs to know about URLs
  - Just more than one
- Gets worse in realistic scenarios
  - Subsections? Promotions? ...
- Need to invalidate based on content
  - Not URL

Non-django changes

- All changes don't come through the webapp
  - Maybe today, but that won't last
- Batch loads
- Direct database edits
- Other applications
  - Direct access vs API

Next step

- All articles have a primary key
  - Surrogate key required by Django
  - Guaranteed by database
- All changes happen in the database
  - Regardless of source

Next step

- Invalidate cache based on primary key
- Invalidate cache from the database

Key based invalidation

- Need to tell cache about articles
- So it can separate them from URLs
- "Surrogate http header"

Modified views

```
@cache(minutes=30)
def article(request, articleid):
    article = get_object_or_404(Article, id=articleid)
    return contains_articles(
        render_to_response('article.html', {
            'article': article,
        }),
        article.pk)
```

Modified views

```
@cache(minutes=5)
def index(request):
    articles = Article.objects.all()
    return contains_articles(
        render_to_response('index.html', {
            'articles': articles,
        }),
        ','.join([str(a.pk) for a in articles]))
```

Header wrapper

```
def contains_articles(resp, articles):
    resp['X-articles'] = articles
    return resp
```

Results

```
mha@mha-laptop:~$ curl -I http://localhost:9000/
```

```
HTTP/1.0 200 OK
Date: Sun, 19 Jan 2014 14:43:52 GMT
Server: WSGIServer/0.1 Python/2.7.3
Content-Type: text/html; charset=utf-8
X-articles: 2,3,1
Cache-Control: s-maxage=300
```

Invalidate from the db

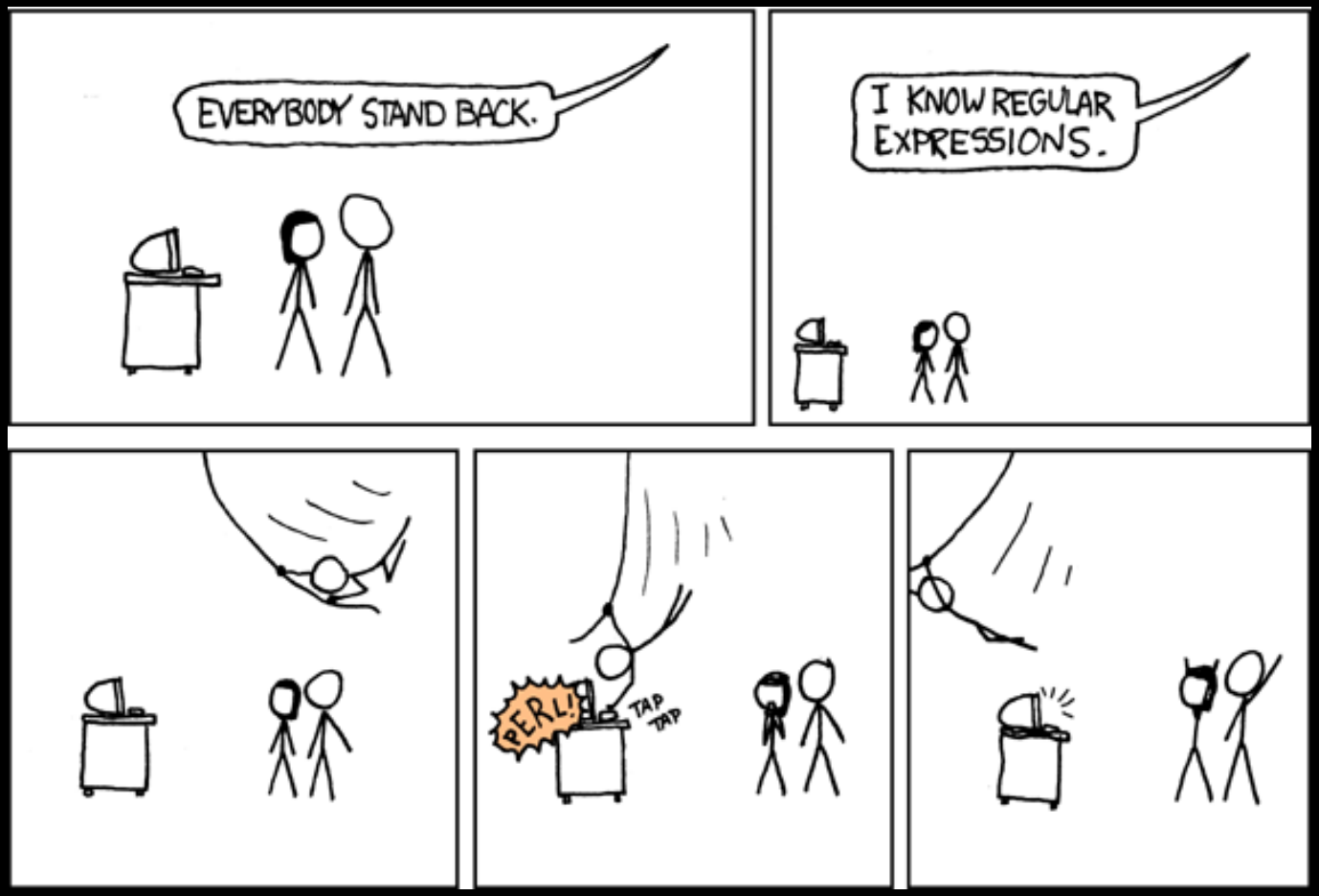
```
CREATE OR REPLACE FUNCTION invalidate_article()
 RETURNS trigger LANGUAGE plpythonu
AS $$
import httplib
id = TD['new']['id']
conn = httplib.HTTPConnection('localhost:9001')
conn.request('PURGE', '/article/%s/' % id)
conn.getresponse()
conn.close()
$$
```

Invalidate from the db

```
CREATE TRIGGER articles_trigger
AFTER UPDATE OR DELETE
ON myshop_article
FOR EACH ROW EXECUTE PROCEDURE
  invalidate_article();
```

Multiple URLs at once

Multiple URLs at once



http://xkcd.com/208/

Combined

```
CREATE OR REPLACE FUNCTION invalidate_article()
 RETURNS trigger LANGUAGE plpythonu
AS $$
import httplib
id = TD['new']['id']
conn = httplib.HTTPConnection('localhost:9001')
conn.request('POST', '/_api/purge', '', {
  'X-articleexpr': '(^|,)%s(,|$)' % id})
conn.getresponse()
conn.close()
$$
```

Combined

```
sub vcl_recv {
  ..
  # Check for our own purge requests
  if (req.url == "/_api/purge" && req.request == "POST") {
    if (!client.ip ~ purge) {
      error 405 "Not allowed.";
    }
    ban("obj.http.x-articles ~ " . req.http.X-articleexpr);
    error 200 "Purged.";
  }
  ..
}
```

Still ugly...

- http calls from inside the database!
- What if the cache is down?
  - Could be a network hiccup?
- Or just very slow?
- Or there is more than one?

Use a queue

- Any message queue
- That delivers to all listeners
- And keeps a backlog

pgq

- Simple PostgreSQL based queue
- Part of skytools
- Independent or cooperative consumers
- SQL API
  - Wrappers in other languages
  - Python, PHP, ...
- Transactional

pgq

- Ticker daemon
  - Makes things happen
  - Must always be running
- Consumers
  - Per queue code
  - Runs when things are posted
  - Grouped batches

Consumer

```
class MyPurger(pgq.Consumer):
  def process_batch(self, db, batch_id, ev_list):
    for ev in ev_list:
      if ev.type == 'A':
        conn = httplib.HTTPConnection('localhost:9001')
        conn.request('POST', '/_api/purge', '', {
          'X-articleexpr': '(^|,)%s(,|$)' % ev.data,
        })
        conn.getresponse()
        conn.close()
```

Simplified trigger

```
CREATE OR REPLACE FUNCTION public.invalidate_article()
 RETURNS trigger LANGUAGE plpgsql
AS $$
  BEGIN
    PERFORM pgq.insert_event('varnish', 'A', NEW.id::text);
    RETURN NEW;
  END
$$
```

Conclusions

- The database is a point of integration
  - Whether you like it or not
  - Use it!
- Surrogate keys are a reality
  - Whether you like it or not
  - Use it!

Conclusions

- Caching will save you at some point
  - Make sure you are prepared
- Doesn't mean you have to give up features
  - Just be prepared
- Don't break layers until you have to

Conclusions

- Smart caching >> naive caching
  - Choose the right tools
  - Use the tools you have!

Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander