Data Storage on Google App Engine and changes to the App Engine pricing model

Kenneth Macleod

Aladdin Schools

kenneth@aladdin.ie

Current Model

- You are charged on:
  - CPU usage, including datastore access.
    - 6.5 hours free
  - Bandwidth
    - 1 GB in / out free
  - Storage Costs
    - 1 GB free

What's wrong?

- Web Apps are IO bound, not CPU bound.
- Current pricing model does not take into account all the resources required to deliver applications, especially memory.
- Google wants to turn App Engine into a sustainable business and exit beta.

New Model, from Nov 1st

- Charged per instance hour
  - 28 instance hours free
  - In Python, one instance can serve 1 request at a time for now
  - Java has multithreading today
- Datastore access is now a chargeable offense
- Min spend of $9 per month
- Other charges remain the same
- Google expects costs to go up 2x - 5x

Instances

- Same idea as an EC2 instance but less powerful and flexible but far easier to manage.
- Google will use their scheduler to spin up and down instances on demand.
- Limited set of controls over when instances are started, no way to hard cap the number.

Controlling Instances

- Make your application faster and tweak the controls

- Extreme case: set Idle instances to 1 and Min Pending Latency to Max.
- May affect users, but better than giving them quota denied errors.
- You can reserve instances which makes them cheaper.

http://point7.wordpress.com/2011/09/04/appengine-tuning-1/ http://goo.gl/rKJbi

Controlling Datastore Costs

- *Write operations (Entity Put, Entity Delete, Index Write)* $0.10 per 100k operations.
- *Read operations (Query, Entity Fetch)* $0.07 per 100k operations.
- *Small operations (Key Fetch, Id Allocation)* $0.01 per 100k operations.

Tips:

- Index as little as possible: use indexed=False
- Use key / id gets instead of fetches
- Memcache is free, use it to reduce cost and increase performance

User Settings Example

```
class User(db.Models):
  name = db.StringProperty()
  def get_setting_value(self, name):
    setting = self.settings.filter('name', name).get()
    if setting:
      return setting.value
    return None

class UserSetting(db.Models):
  user = db.ReferenceProperty(User, collection_name='settings', required=True)
  name = db.StringProperty()
  value = db.StringProperty()
```

- Several problems, we're using a fetch to get the settings
- We're not using memcache
- Things could slip and we end up with 2 settings

User Settings Improved

```
def get_user_setting_key(user, name):
  return '%s_%s' % (user.key().id(), name)

class User(db.Models):
  name = db.StringProperty()
  def get_setting_value(self, name):
    key_name = get_user_setting_key(self, name)
    mem_key = 'usersetting_%s' % key_name
    setting = memcache.get(key_name)
    if not setting:
      setting = UserSetting.get_by_key_name(key_name)
      if setting:
        memcache.set(key_name, setting.value)
    if setting:
      return setting.value
    return None

class UserSetting(db.Models):
  @classmethod
  def create_new(cls, user, name, *args, **kwargs):
    key_name = get_user_setting_key(user, name)
    return cls(key_name=key_name, *args, **kwargs)
  value = db.StringProperty(indexed=False)
```

Get from memcache first, free and faster

Get by key name rather than query, cheaper and faster

Put it into memcache for next time

No need for name and user as properties in the class, no need to index value

Is it still worth it?

- Yes, based on TCO, especially ease of use and scalability. You don't need a member of staff dedicated to keeping your app running and patched.
- There are much cheaper options, but they don't give you as much so easily.
- Lock-in is the 800 pound gorilla, you have to trust Google, and the pricing changes have dented that for some people.
- Google moving App Engine out of beta makes it worthwhile, there is a long term future for App Engine.