

The 90-Day Startup on Google AppEngine

David Chandler

dchandler@TurboManage.com

<http://TurboManage.com>

Who is this guy?



Wrote first book on running a Web site (1995)

Written Web apps in perl, ksh, C with lex/yacc, ColdFusion, JSF, GWT

Most recently, Web architect with Intuit on Internet banking products

Patent on non-recursive SQL trees

Working in a startup last 6 mos

AppEngine for Java

What is it?

Getting started

Where does it fit?

Designing for AppEngine

An architecture that works

Unsolved problems

Resources

What is AppEngine?

Business: Google's cloud computing platform competing with Azure and AWS

Technical: giant Web farm in front of world's largest HashMap

Java developer: easiest and most scalable Java hosting environment

Startup developer: FREE to get started

Enterprise developer: tools that make you drool, but Security will never allow it



Let's do it

Sign up for AppEngine beta

Download Google plugin for Eclipse

Create new Google Web application

Deploy it

What didn't we do?

Install Tomcat

Configure Tomcat

Start Tomcat

Stop Tomcat

Install a database

Configure a database

Find a DBA to run scripts to populate a database...



What didn't we do?

But what would we do differently to SCALE

our Web tier?

our data tier?

our asynchronous service tier?

For example,

Azure: provision more SQL or IIS instances in admin console

AWS: same, using Rightscale or similar

AppEngine: see checklist...



Tradeoffs

You have to build the AppEngine way

- BigTable is not a relational database
- Can't start your own threads
- JRE whitelist
- Don't forget quotas

Working with the Datastore

Hierarchical, not relational

Entity may have child entities

Entity + children = entity group

Transactions only work within entity group

But deeply nested entity graph limits perf

Datastore can only write 5-10 updates / sec per entity group (instance, not class, thankfully)

To update many entities in one request, they must be root entities

Datastore APIs

Native

get(Key key), put(Entity e), etc.

But...entities are not POJOs, so you need...

JDO or JPA

JDO supported first, seems to be Google's favorite

JPA well-supported now, Google is committed

Objectify (<http://code.google.com/p/objectify-appengine>)

Twig, SimpleDS, others

Datastore limitations

Some RDBMS features can be implemented
in JDO / JPA drivers

- Lazy fetching (be careful)

- Cascading deletes

- Relationship modeling with Collections

Others can't

- Datastore doesn't enforce FK relationships

- Query restrictions

Query restrictions

No OR clauses (requires separate trips to the Datastore)

IN supported, but makes separate trips

Inequality filters (<, >, !=) on one prop only

Joins barely supported

As of Feb 12, can do 2-way == joins with JDO or JPA only

What IS supported?

Fast, scalable index retrieval

Remember, it's a HashMap

Define multi-property indexes in datastore-
indexes.xml

JDO + JPA auto-create them for you

Merge-join

AND clause on multiple properties

AppEngine uses efficient zig-zag algorithm to
satisfy AND conditions without multi-prop idx

Datastore vs. RDBMS

Banking application

Forget it.

You need transactions, enforced relationships

Social networking application

Absolutely! Latency, asynchronous writes OK

List properties solve the million-fan-out problem

General business app (issue tracker)

Tx not so important, can solve for latency

What are list properties?

@Entity

```
public class PrayerIndex
```

```
{
```

```
    @Id
```

```
    private Long id;
```

```
    @Parent
```

```
    private Key<PrayerItem> parent;
```

```
    private List<Key<User>> followerKeys;
```

Whoa--isn't that bad design?

Rules of normalization disallow repeating elements in a “column”

Normally use intersection table to model a many-to-many relationship

In fact, that's what Datastore is doing

Creates index row for each item in the list

So you can do this:

```
indexQuery.filter("followerKeys", followerKey);
```

Joins revisited

Query on list property can be used in place of a join with an intersection table

Some things are much easier this way

Example: find mutual friends of Joe and Sam

SQL:

```
SELECT FROM Friends a JOIN Friends b ON  
a.friend = b.friend WHERE...
```

AppEngine:

```
filter("friends", "Joe").filter("friends", "Sam")
```

Datastore specialties

List properties (Google I/O '09, Brett Slatkin)

Keys-only queries (avoids deserialization)

Efficient batch gets: `get(keys)`

And puts: `put(entities)`

Ancestor queries--find all child entities

Web-safe cursor!

<http://gae-java-persistence.blogspot.com/>



Why Objectify?

Light

36k jar, no external dependencies

Fast

no bytecode enhancement on start

Simple

No PersistenceManager lifecycle means no DI required, no attach / detach to mess with

@Cache (uses AppEngine Memcache)

@Embedded (objects as list tuples)

How to do background processing?

Your application can't start threads

Causes problems for some frameworks

See [will-it-play-in-appengine](#) wiki

No, you can't run WebSphere or JBoss on AppEngine

AppEngine provides Task Queue API

Servlet-based

Allows throttling

Queuing a Task

Define queue in WEB-INF/queue.xml

```
<queue>  
  <name>mail</name>  
  <rate>8/m</rate>  
</queue>
```

Create worker servlet that handles task

Queue the task

```
queue.add(url("/mail").param("key", key)...) )
```



Is there an easier way?

```
public class HelloUserTask implements Deferrable
{
    private User user;

    public HelloUserTask(User u)
    {
        this.user = u;
    }

    @Override
    public void doTask() throws ServletException, IOException
    {
        System.out.println("Hello, " + user.getNickname());
    }
}
```

Is there an easier way?

Vince Bonfanti's Deferred task servlet

```
HelloUserTask task = new HelloUserTask(testUser);  
// Queue the task  
Deferred.defer(task);
```

Task throttling can help with quotas

An architecture that works

Objectify

Guice

Gwt-dispatch (GWT-RPC)

QueuingBatchingCachingDispatchAsync



Guice

Google's dependency injection framework

Small, light, almost plain English

See

ServerModule

DispatchServletModule

DispatchTestModule

ROAUserServiceImpl

Used by gwt-dispatch, useful for unit tests



gwt-dispatch

GWT-RPC basics: every service = 3 classes

Service interface (LoginService)

Async version (LoginServiceAsync)

Server impl (LoginServiceImpl)

```
@RemoteServiceRelativePath("gwtlogin")
```

```
public interface LoginService extends RemoteService
```

```
{
```

```
    public LoginInfo login(String requestUri) throws  
        UserNotRegisteredException;
```

```
}
```

gwt-dispatch

Eclipse tooling auto-creates the asyncs

What if you wanted to add a standard arg to every service call?

Like a security token to prevent CSRF

Would have to add to every interface, async, and impl UGH

Or what if you wanted to cache data from service calls?

gwt-dispatch

Turn verbs (methods) into nouns (classes)

```
loginService.login(p1,p2...,callback);  
dispatchService.execute(new LoginAction(),  
    new AsyncCallback<LoginResult>()  
    {  
        handleSuccess(LoginResult result)...  
        handleFailure(Throwable caught)...  
    })
```



gwt-dispatch

Command pattern benefits

Send security token in DispatchService and
Actions don't even have to know about it

Actions and ActionHandlers can extend base
classes

Constructor params are saved with the cmd

Caching! Batching! Queuing!

Rollback!

Unsolved problems

Filter Logs [?](#)

Minimum Severity: [+ Options](#)

Tip: Click a log line to show or hide its details.

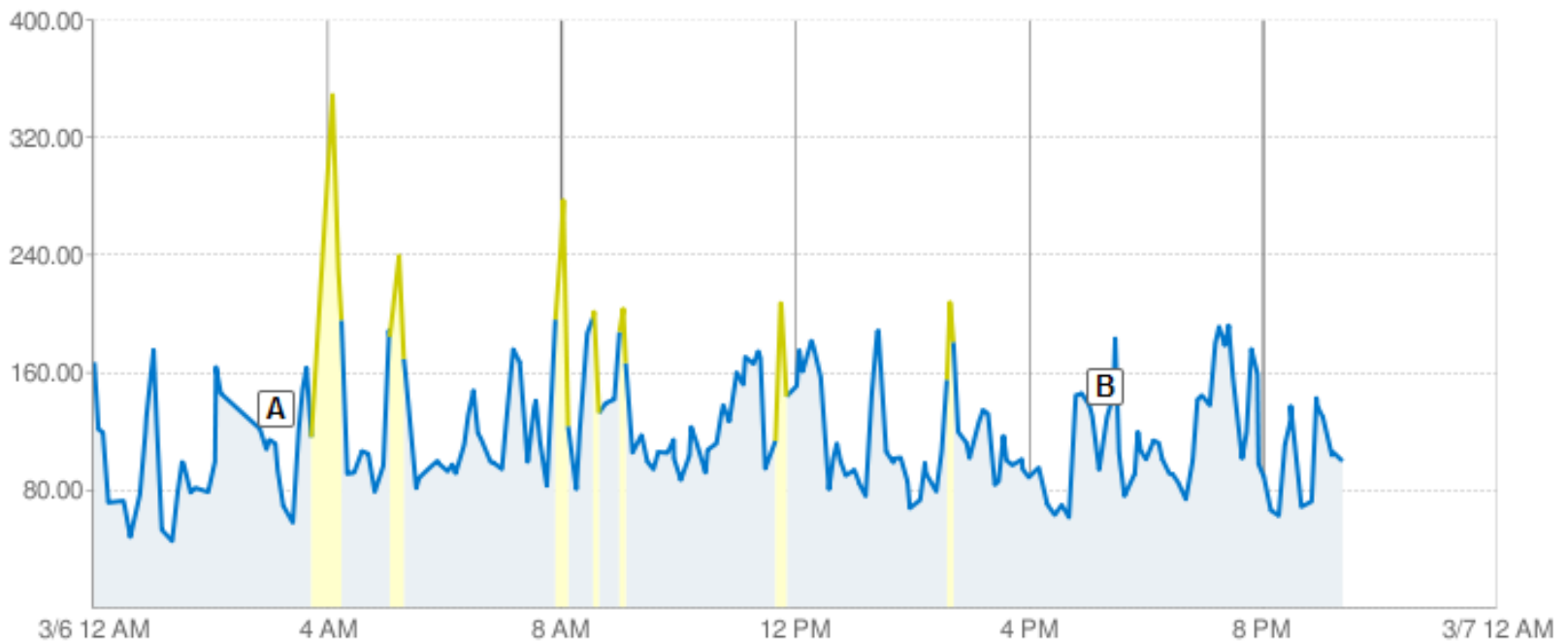
◀ Prev 20 **1-20** [Next 20](#) ▶

+	03-05 06:43PM 40.967	/roa/8B5C3A0D9A73F6725A5918C7909AC476.cache.png	200	7ms	0cpu_ms	0kb	Mozilla/5.0 (Windows; U; Wind
+	03-05 06:43PM 31.757	/roa/dispatch	200	9127ms	7847cpu_ms	30api_cpu_ms	⚠ 0kb Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US
+	03-05 06:30PM 05.198	/roa/dispatch	200	6101ms	8243cpu_ms	76api_cpu_ms	⚠ 0kb Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US
+	03-05 06:25PM 42.624	/roa/dispatch	200	2149ms	138cpu_ms	21api_cpu_ms	0kb Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1
+	03-05 06:25PM 42.574	/roa/dispatch	200	2059ms	1137cpu_ms	126api_cpu_ms	⚠ 0kb Mozilla/5.0 (Windows; U; Windows NT 6.0; en-U
+	03-05 06:25PM 43.047	/roa/images/tick_24.png	200	6ms	0cpu_ms	0kb	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.1.8) Ge
+	03-05 06:25PM 42.521	/roa/dispatch	200	460ms	322cpu_ms	30api_cpu_ms	0kb Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9
+	03-05 06:25PM 42.739	/roa/gwt/standard/images/hborder.png	200	7ms	0cpu_ms	0kb	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-U
+	03-05 06:25PM 42.682	/roa/images/cross_16.png	200	5ms	0cpu_ms	0kb	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.1.8) C
+	03-05 06:25PM 42.537	/roa/02A4E10B4D1783A5F108EF83FCEAA7CF.cache.png	200	6ms	0cpu_ms	0kb	Mozilla/5.0 (Windows; U; Winc
+	03-05 06:25PM 42.477	/roa/clear.cache.gif	200	9ms	0cpu_ms	0kb	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.1.8) Gecko/

Unsolved problems

Get: Latency

Measures the latency, in milliseconds, of one call to `datastore.get()` on a 2kB entity selected at random from a predefined list.



One more thing

AppEngine for Java < 1 yr old

gwt-presenter, gwt-dispatch < 5 mos

Many Datastore features < 2 mos

Objectify barely 1 mo

Bleeding edge: $(t_{\text{problem}} - t_{\text{solution posted}}) == 0$

How close can you tolerate?

How long did it really take?

Deploy sample app	90s
Ooh & aah at admin console	90m
Wait for frameworks I needed	90d
Learn how to use them	30d
Write application code	60d
Upgrade them constantly	90h

Resources

AppEngine docs (a little thin, but good)

Forums

<http://groups.google.com/group/google-appengine-java>

(Google engineers monitor)

<http://groups.google.com/group/google-appengine-downtime-notify>

<http://gae-java-persistence.blogspot.com/>

<http://turbomanage.wordpress.com> (Objectify, gwt-dispatch, gwt-presenter example code)

Upcoming Google I/O conf