Google AppEngine - A Second Look

Saturday, February 21, 2009 at 12:56AM

Todd Hoff in AWS, GAE, Python

Update 6:: Back to the Future for Data Storage. *We are in the middle of a renaissance in data storage with the application of many new ideas and techniques; there's huge potential for breaking out of thinking about data storage in just one way.*
Update 5: Building Scalable Web Applications with Google App Engine by Brett Slatkin.

Update 4: Why Google App Engine is broken and what Google must do to fix it by Aral Balkan. *We don't care that it can scale. We care that it does scale. And that it scales when you need it the most.* Issues: 1MB limit on data structures; 1MB limit on data structures; the short-term high CPU quota; quotas in general; Admin? What's that?
Update 3: BigTable Blues. Catherine Devlin couldn't port an application to GAE because it can't do basic filtering and can't search 5,000 records without timing out: "Querying from 5000 records - too much for the mighty BigTable, apparently." Followup: not the future database. "90% of the work of this project has been trying to figure out workarounds and kludges for its bizzare limitations."
Update 2: Having doubts about AppEngine. Excellent and surprisingly civil debate on if GAE is a viable delivery platform for real applications. Concerns swirl over poor performance, lack of a roadmap, perpetual beta status, poor support, and a quota system as torture chamber model of scalability. GAE is obviously part of Google's grand plan (browser, gears, android, etc) to emasculate Microsoft, so the future looks bright, but is GAE a good choice now?
Update: Here are a few experience reports of developers using GAE.

Diwaker Gupta likes how easy it is to get started on the good documentation. Doesn't like all the limits and poor performance. James here and here also likes the ease of use but finds the data model takes some getting used to and is concerned the API limits won't scale for a real site. He doesn't like how external connections are handled and wants a database where the schema is easier to manage. These posts mirror some of my own concerns. GAE is scalable for Google, but it may not be scalable for my application.

It's been a few days now since GAE (Google App Engine) was released and we had our First Look. It's high time for a retrospective. Too soon? Hey, this is Internet time baby. So how is GAE doing? I did get an invite so hopefully I'll have a more experience grounded take a little later. I don't know Python and being the more methodical type it may take me a while. To perform our retrospective we'll take a look at the three sources of information available to us: actual applications in the AppGallery, blogspew, and developer issues in the forum.

The result: a cautious thumbs up. The biggest issue so far seems to be the **change in mindset needed by developers to use GAE**. BigTable is not MySQL. The runtime environment is not a VM. A service based approach is not the same as using libraries. A scalable architecture is not the same as one based on optimizing speed. A different approach is needed, but as of yet Google doesn't give you all the tools you need to fully embrace the red pill vision.

I think this quote by Brandon Smith in a thread on how to best implement sessions in GAE nicely sums up the new perspective:

> *Consider the lack of your daddy's sessions a feature. It's what will make your app scale on Google's infrastructure.*

In other words: when in Rome. But how do we know what the Romans do when the Romans do what they do?

Brett Morgan expands our cultural education in a thread on slow GAE databases performance when he talks about why MySQL thinking won't work on BigTable:

> *It might look almost look like a sql db when you squint, but it's optimized for a totally different goal. If you think that each different entity you retrieve could be retrieving a different disk block from a different machine in the cluster, then suddenly things start to make sense. avg() over a column in a sql server makes sense, because the disk accesses are pulling blocks in a row from the same disk (hopefully), or even better, all from the same ram on the one computer. With DataStore, which is built on top of BigTable, which is built on top of GFS, there ain't no such promise. Each entity in DataStore is quite possibly a different file in gfs.*
>
> *So if you build things such that web requests are only ever pulling a single entity from DataStore - by always precomputing everything - then your app will fly on all the read requests. In fact, if*

> *that*
> *single entity gets hot - is highly utilized across the cluster - then*
> *it will be replicated across the cluster.*
>
> *Yes, this means that* ***everything that we think we know about building***
> ***web applications is suddenly wrong****. But this is actually a good thing.*
> *Having been on the wrong side of trying to scale up web app code, I*
> *can honestly say it is better to push the requirements of scaling into*
> *the face of us developers so that we do the right thing from the*
> *beginning. It's easier to solve the issues at the start, than try and*
> *retrofit hacks at the end of the development cycle.*

A truly excellent explanation of the differences between MySQL thinking and GAE thinking.

Now, if you can't use MySQL's avg feature, how can an average be calculated using BigTable? Brett advises:

> *Instead of calculating the results at query time, calculate them when*
> *you are adding the records. This means that displaying the results is*
> *just a lookup, and that the calculation costs are amortized over each*
> *record addition.*

Clearly this is more work for the programmer and at first blush doesn't seem worth the effort, especially when you are used to the convenience of MySQL. That's why in the same thread Barry Hunter insightfully comments that GAE may not be for everyone:

> *This might be a very naive observation, but I perhaps wonder then if*
> *GAE is the tool for you.*
>
> *As I see it the App Engine is for applications that are meant to*
> *scale, scale and really scale. Sounds like an application with a few*
> *hundred hits daily could easily run on traditional hosting platforms.*
>
> *It's a completely different mindset.*
> *...*
> *Again maybe I am missing something, but* ***the DataStore isn't designed to be super fast at the small scale, but rather handle large amounts of data, and be distributed*** *(and because its distributed it can appear very fast at large scale).*
>
> *So you break down your database access into very simple processes.*
> *Assume your database access is VERY slow, and rethink how you do things. (Of course the piece in the puzzle 'we' are missing is*

MapReduce! - the 'processing' part of the BigTable mindset)

Before developers can take full advantage of GAE these types of lessons need to be extracted and popularized with the same ferocity the multi-tier RDBMS framework has been marketed. It will be a long difficult transition.

Interestingly, many **lessons from AWS are not transferable to GAE**. AWS has a VM model whereas GAE has an application centric model. They are inverses of each other.

In AWS you have a bag of lowish level components out of which you architect your application. You can write all the fine low level implementations bits you desire. A service layer is then put in front of everything to hide the components. In GAE you have a high level application component and you build out your application using services. You can't build any low level components in GAE. In AWS the goal is to drive load to the CPU because CPU and bandwidth are plentiful. In GAE you get very limitted CPU, certainly none to burn on useless activities like summing up an average over a whole slice of data returned from SimpleDB. And in GAE the amount of data returnable from the database is small so your architecture needs to be very smart about how data is stored and accessed.

Very different approaches that lead to very different applications.

Applications

The number of applications has exploded. I am always amazed at how enthusiastic and productive people can be when they are actually interested in what they are doing. It happens so rarely. True, most

applications aren't even up to Facebook standards yet, but it's early days. What's impressive is how fast they were created and deployed. That speaks volumes about the efficacy of the application centric development model.Will it be as effective delivering "real" apps? That's a question I'm not sure about.

So far application performance is acceptable. Certainly nothing spectacular. What can you do about it? Nada.

I like the sketch application because people immediately and quite predictably drew lewd depictions of various body parts. I also like this early incarnation of a forum app. A forum is one of the ideas I thought might work well on AppEngine because the scalable storage problem is solved. I do wonder how the performance will be with a fine tuned caching layer? Vorby is a movie quote site showing a more realistic level of complexity. It has tabs, long lists of text, some graphical elements, some more complex screens, and ratings. It shows you can make applications you wouldn't mind people using.

An option I'd like to see in the App Gallery is a *view source* link. Developers could indicate when adding an application if others can view their application source. Then when browsing the gallery we could all learn by looking at real working code. This is how html spread so quickly. Anyone could view the source for any page, copy paste, and you're on your way! With an application centric model the view source viral spread approach would also work.

Blogspew

As expected there's lots of blog activity on GAE:

- As to all those people complaining their favorite language isn't available, take a chill pill, Urubatan asks us When will programmers

learn that a language is just a tool?. I mostly agree with this take, but I also agree with a commenter who observed that it's a lot harder for a team of developers to turn on a dime and adopt a whole new everything.

- Garrick Van Buren says Free & Open Is Its Own Lock-in. The idea being it's worthing paying something you know works, allows you to experiment, and you are aligned with their zeitgeist. Leaving that for "free" isn't a good deal.
- evan_tech: google app engine limitations. Don't focus on minor problems. The big problems are: all code runs only in response to HTTP fetches, No long connections means no "comet" (server-push messaging), playing around with your data is hard as there's no way to perform operations on your data except by uploading code to the server, Table scans are slow and you can't cache because it's so slow you hit your CPU limit, bulk operations are hard, and no arbitrary queries.
- RedMonk Clouds Rolling In: The Google App Engine Q&A gives covers a lot of GAE territory. List some of the cons: Python only, not database export, lock-in, and no cron. "...all of the current offerings have limitations that throttle their usage. Many of which are related to the lack of open standards. Apart from the mostly standard Python implementation, App Engine is decidedly non-standard."
- Alex Bosworth pits AWS vs Google App Engine in a death match. Alex thinks: To be succinct, based on where the Google App Engine is today, I would say AWS still has a strong lead in application hosting, and I would not currently consider writing an application for Google’s current platform. Cons: Lockin, The page-view limitation is quite low, no memcache, No long running pages, or cron jobs, Storage size limitation, One language, No requests unless they are through Google’s API. Pros: it’s free, looks pretty rocking, integrates with Google accounts.
- Joyent is countering by offering free infrastructure for high volume python applications. Joyent only asks "that you provide Joyent unlimited access to your customer information and clickstream data." Your data has a lot of value. Google is also very aware of that. More in my Why Does

Google Do What Google Does? post. Though the Joyent's building blocks approach is very different than Google's application centric approach. We'll see which matters more: the model or facilities?

- Niall Kennedy in Google App Engine for developers does a great job contrasting the complexity of your normal website setup with an application approach. Normally you: purchase dedicated servers or virtualized slices, capacity plan, configure web server, install Python, Apache, setup MySQL in scalable fault tolerant configuration, insert caching layer, add monitoring layer, add static file serving and bulk file serving, make it all work together, spend your life keeping it working and responding to failures. Nicely drawn contrast to upload and go.
- TechCrunch's AppEngine test application couldn't handle a TechCrunch level of load, which is a little concerning. This means usage limits are set a bit low and with no pricing model to work from it's reasonable to be concerned about the cost. Nobody wants a cell phone overage nightmare for their website costs.
- Groovy: Google Datastore and the shift from a RDBMS. An excellent comparison of how BigTable differs from a RDBMS. The conclusion: The end result of this, is that the standard way a developer writes out the table schema for a RDBMS should be dumped almost entirely when considering an app using Google Datastore.
- Service Level Automation in the Datacenter: What Google App Engine is NOT. It's a web play only, it's not a cloud in the sense of datacenter infrastructure IT can move to. You can't implement: Portal Services, SOA architectures, Business Process Automation, Enterprise integration, HPC, and Server and desktop virtualization.

A lot has been made of the risk of lock-in. I don't really agree with this as everything is based around services, which you can port to another infrastructure. What's more the problem is developers will be acquiring a sort of **learned helplessness**. It's not that developers can't port to another environment, they simply won't know how to anymore because they will

have never had to do it themselves. Their system design and infrastructure muscles will have atrophied so much from disuse that they'll no longer be able to walk without the aide of their Google crutches. More in another post.

Developer Forum

The best way to figure out how a system is doing is to read the developer support forum. What problems and successes are real developers experiencing trying to get real work done? The forum is a hoppin'. As of this writing over 1300 developers have registered and nearly 400 topics are active. What are developers talking about?

- Please support my favorite language: PHP, Ruby, etc. Hey, they had to start somewhere and Python is as good as anything else. A language is just a tool you know :-)
- The usual this doesn't work in my environment type of questions. Far fewer than I would expect though.
- The switch away from RDBMS thinking isn't coming naturally. A lot of questions wondering how to access BigTable like MySQL and that won't work. There are no joins in GQL, so how do you do normal things like get all the comments for a blog post?
- Lots of how do use this or that API questions. Lack of certain commonly used APIs, like XML parsers is being being encountered.
- Concern there's no database export. You can bulk upload data, but you have to write your own program to get it out again.
- People are hitting limits like the 1MB upload limit on all requests. The 1000 database return limit is mentioned a lot. This is very different than the AWS model which advocates moving work to your CPU so it makes sense to return large sets of data. Google limits your CPU usage and the amount of data you can return so you have to be smart how you store and query data.
- The pure service model has profound limitations for certain application

types. An issue of how to do image processing came up. Usually a compiled class is used because using pure Python is slow. But you can't load these types of classes in AppEngine. And you can't parallelize the work by farming it out to other CPUs. You are stuck. Here's were a .Net type managed object model might help.

- Surprisingly, fulltext search is not supported.
- Sessions are another how do I it on GAE question. People are used to frameworks handling session storage.
- One user was surprised at how slow database access was with BigTable. It takes GAE almost 3 seconds to save 50 of dummy records (consisting of just 2 text fields). A nice thread about how best to use BigTable developed. BigTable is meant to scale and you have to do things differently than you do in a MySQL world.

Many "how do I" questions come up because of the requirement for service level interfaces. For example, something as simple as a hostname to IP mapping can't be done because you don't have socket level access. Someone, somewhere must make a service out of it. Make an external service is a common response to problems. You must make a service external to the GAE environment to get things to work which means you have to develop in multiple environments. This sort of sucks. To get cron functionality do I really need to create an external service outside of GAE?

The outcome of all this is probably an **accelerated servicifaction** of everything. What were once simple library calls must now be exposed with service level interfaces. It's not that I think HTTP is too heavy, but as development model it is extremely painful. You are constantly hitting road blocks instead of getting stuff done.

Article originally appeared on High Scalability (http://highscalability.com/).