What Google App Engine Price Changes Say About the Future of Web Architecture

Wednesday, September 7, 2011 at 8:32AM

Todd Hoff

When I was a child, I spake as a child, I understood as a child, I thought as a child: but when I became a man, I put away childish things. -- Corinthians

With this new pricing, developments will be driven by the costs. I like to optimize my apps to make them better or faster, but to optimize them just to make them cheaper is a waste of time. -- Sylvain on Google Groups

The dream is dead. Google App Engine's bold *pay for what you use* dream dies as it leaves childish things behind and becomes a real product. Pricing will change. Architectures will change. Customers will change. Hearts and minds will change. But Google App Engine will survive.

Google is shutting down many of its projects. GAE is not among them. Do we have GAE's pricing change to thank for it surving the more wood behind more deadly arrows push? Without a radical and quick shift towards profitably GAE would no doubt be a historical footnote in the long scroll of good ideas. The urgency involved is clearly reflected in GAE's offering a 50% pricing discount and moving to the new pricing scheme before the multi-threaded version of Python has been rolled out.

The dream was beautiful: **pay for what you use and make it dead easy**

to use. Now that's an easy vision to understand. It's fair and compelling. GAE had 3 years of a trial run to live the dream, and for the bottom line, it was apparently a nightmare. That's the shame of it. Google was bold. They created something new. They worked hard. GAE revolutionised the development of scalable applications and has extended PaaS in original and interesting ways. They did an amazing job. And the result was well liked, even loved by many. But the economics failed them and they've had to pivot. Hard. What is the result of that change?

Pay for what you use has changed. From an abstract resource driven model, where pricing is pegged to actual CPU usage, GAE is moving to an instance driven model, in the Amazon style, where pricing is pegged to the fully burdened cost of real physical assets (see the FAQ for details). Estimates range from between 2x to 10+x cost increases for running on GAE with the new pricing scheme.

Dead easy has changed. Pricing is just part of the story. GAE still delivers on its zero platform maintenance pledge, but much more pressure is being put on programmers to navigate the new pricing model and recode to a complex moving target in order to minimize costs. It's no longer that simple to use. Work has shifted to clever programming.

Everyone will pay more. And that's not wrong. GAE has every right to make money. It's the speed of the change, the direction of the change, and the magnitude of the change that has thrown people for a loop. Selecting GAE was a leap of faith, and much of the anger is because many feel that faith has been betrayed.

Why Change? Going Enterprise

Wesley Chun, from Google's developer relations, does a solid job trying to explain the reasoning behind the change (summarized):

- GAE is a **premium service**, providing massive scalability for low effort. That's difficult to get elsewhere in the market. GAE serves more than 1.5 billion pages views a day across all applications.
    - Using GAE you only have to worry about your application. On EC2, for example, you have to worry about everything: elasticity/scale, OS, databases, web servers, load balancers, license, patches, upgrades, and so on.
    - Scalability is the most difficult and most expensive thing to build on your own. Using GAE your app can scale to mind blowing numbers, see: Demi Moore and the Royal Wedding.
- GAE as a distributed application platform is a high cost service. Google has spent many years and lots of money to build infrastructure that is Google-scale. With GAE Google is letting you leverage all the research and horsepower.
- A SLA and paid support are being added.
- GAE cannot continue to operate the services at a loss. The service costs significant resources in hardware and networking bandwidth.
- Rather than being cancelled GAE is being released as an official product.
- It's time to pay.
- Mind blowing scalability and ease of use are the reasons to use App Engine. It's not a cheap hosting service. Go elsewhere for that.

All very sensible. As a premium enterprise level product, pricing doesn't have to follow costs, prices can rise to meet competition, or even go higher if you think your offering has particular value.

Yet that's not how GAE started. GAE started as a reasonable cost, easy to use platform for the every programmer, that was billed based on CPU usage. That's all gone now. And if this approach didn't work then it should have been obvious earlier, so it's hard to shake that there's a bait-and-switch feel to this.

The switch to a premium branding is especially surprising because it was never marketed as a premium service. From a business perspective that makes sense now, but that's part of why there's such furor over all this.

What GAE now wants to be is an **enterprise class service.** Money made on other products will pay for a site, it no longer targets developers who can use the lower cost structure of GAE to make money on the applications hosted on GAE itself. Many app sizes, app types, and businesses models won't fit on GAE anymore.

That's a really sharp pivot. But that's what survival makes you do. Really hard choices have to be made. After we've gone through all the stages of grief, we have to get over it, reevaluate, and move on.

mdasen nicely captured the heady feeling of these early days:

> *Hey hackers! You should totally rewrite your apps for our Google systems that are a lot more efficient than other systems. Yeah, there are some annoying restrictions that you'll have to get used to and are totally a pain for some things. Still, out service is cheap for loads of usage and really cheap even after that so you're spending a little programmer time for no-hassle-scaling and cheaper hosting than anything you can get!*

With equal poignancy Stephen, in this Google Groups thread, captures the feeling of today:

> *The situation is that 'App Engine for Business' has been*

renamed 'App Engine' and 'App Engine' as we knew it does not exist any more. They've been pretty clear about that. App Engine is an Enterprise product now and price has no relation to cost.

A Dream Deferred

We can look back now and consider this time a period of experiment...an experiment that has failed in one key aspect: the **resource based pay as you go model does not seem to work**.

Part of the shock is that prices rose at all. We normally think of compute resources getting cheaper over time. Moore's law combined with the economies of scale have been our good friends. Scalability is flowing in a reverse direction, Google is asking users for budget scalability, and developers weren't expecting that.

But this issue isn't one of just profit and loss, the result of this experiment has some interesting implications for the future of application architectures. A topic we'll now explore.

I was totally wrong. I thought GAE Was Going Task Queues

To continue my poor prediction production, GAE's move to instances is the exact opposite of where I thought they were going. I thought GAE would completely lose the instance image concept all together in favor of applications being written on one giant task queue container.

The basis of this conjecture/vision is the development and evolution of one of GAE's most innovative and far reaching features: task queues. It's a great feature that allows applications to be decomposed into asynchronous flows. Work is queued and executed at some later time.

Instead the monolithic and synchronous model used originally by GAE, an application can be completely asynchronous and can be run on any set of machines. For a while now it has been clear the monolithic front-end instances have become redundant with the fruition of task queues.

The problem is task queues are still image based. Operation are specified by a URL that terminate inside a run time instance whose code template is read from an image. An image contains all the code an application can execute. It's monolithic.

When a web client URL is invoked it executes code inside a monolithic image. It's these large images that must be managed by GAE and why Google needs to charge you more. They take resources to manage, time to initialize, and while running take memory even if your app isn't doing anything.

A different idea is to ask why can't a request terminate at a task queue work item instead? Then the monolithic image could be dropped infavor of an asynchronous coding model. Yes, GAE would have to still manage and distribute these code libraries in some fantastical way, no simple task, but this would solve the matching work to resources granularity problem that they instead solved by going the other direction, that is making images the unit of distribution and instances the unit of execution. We'll talk more about the granularity problem next.

So with this super cool task queue framework and programming model being developed I felt sure they were ready to announce that the monolithic images would disappear, instances would disappear, and there would be an even finer pay for what you use billing model as a replacement. I was wrong. Again.

It's a Problem of Quanta Mechanics

Driving this upheaval is that programs run on an abstract machine that uses resources that are quantized differently than the underlying physical machines. A server comes with only so much memory and CPU. Running programs use memory even when a program is idle. Google must pay for the machine resources used by an instance. Charging only for the resources used by a program instead of all the resources used to host a program creates an unsustainable and unprofitable pricing friction between the two models.

In other words, programs are deployed in big quanta, but run in small quanta. A smaller work granularity would allow work to be schedule in idle times, which is why I think the task queue model is superior.

We see this effect in practice. An application that used .02cpu hours may now take 2.8 instance hours. If you coded your application correctly in the past, when GAE looked like one giant CPU, as few CPU resources as possible were used. Now GAE has exploded into a set of components (Frontend, backend, scheduler, etc).

Minimizing CPU doesn't matter as much now. If an application blocks on IO you are being charged because you have an instance running. That instance costs money to run so you are paying for it no matter what. That instance takes time to startup up so you pay for 15 minutes regardless of how long you need it for. And if a request will take too long in the scheduler's eyes, more instances will be spun up. You can't really argue with that I think. The key is to change how resources are shared. Punting to an instance model was kind of giving up.

Many people have asked for a separate resource charges for memory if Google thought memory a scarce resource. This would put pressure on applications to use memory efficiently. The problem is memory and CPU aren't separable with an instance based approach. This is also why Google can't make a perfectly efficient scheduler, the units of work don't match up to the compute resource units.

Costs are Rising

- Raymond C: Monthly charge: $900 -> $2850 (310%)
- Daniel: I'm seeing an increase of 300%.
- joakime: Our costs went up 2800% under the new pricing.

Costs are rising because instance hours are increasing over CPU based billing, which multithreaded Python will help fix, but there are also cost increases in datastore operations.

The Amazing Story Of AppEngine And The Two Orders Of Magnitude

Emlyn O'Regan has written a really fine article explaining in great detail his experiences tuning his application for the new pricing model. He has also followed this article up with AppEngine Tuning #1.

At first, like most, he ignored the original announcement thinking the changes wouldn't be big, but they were. His bill went from $0.51/day to $49.92/day. Instead of paying $200 a year it would be $20K a year. A huge jump.

What Emlyn does next is what's cool, he takes us through a tour of his app, finding where the money is being spent, and how he changed his app to reduce costs using Google tools and documentation. Under the new model we find some practices that weren't a problem in the past, but are now.

The issues:

1. **Frontend Instance Hours.** His app went from 11.39 CPU Hours per day to 231.13 Instance hours per day. The problem turned out to be that a lot of parallel work was created. This caused the scheduler to create more instances to handle the work. The increase in instances caused a lot more instance hours to be used compared to what the work would seem to entail. No fix for this one yet, but the problem is understood now. For work like this, background work where latency doesn't matter, the scheduler should be way less aggressive.
2. **Datastore Reads**. This turned out to be the big cost hit. It turns out his app was doing 60 Million operations on the datastore per day. First, he was keeping old data around, which was being read during table scans. That costs. It turns out by using the fetch call and paging through records using limit and offset, you can really crank up the number of datastore reads. Designs need to think about how to minimize the number of reads. A code fix could remove the need for this path in the code or it could be moved to a backend operation and run once per day instead of the every two minutes it was running.
3. **Tuning the scheduler is effective.** Setting the Max Idle Instances to 1, and Min Pending Latency to 15 dropped the instance average to 4 or 5 instances rather than 10 to 15, with no decrease in perceived performance.

Some issues resulted from just make it work type design. Typical in a project. GAE's new pricing mechanism now makes this kind of laxity **costly so it will get fixed**. A good thing. Architectures will have to think about these issues from the start now. **Minimize instances and minimize database touching**. Check.

For a GAE what this will result in is less overall resource usage, which is what they want. Sending appropriate pricing signals is the most efficient way of achieving that result.

Other problems were because of unexpected consequences of innocent looking GAE API calls. Consequences should be more obvious in the API. It will take a **while to work out** all this out because it is so subtle and app specific.

Scheduler as a Conflict of Interest

The scheduler's job is to schedule work across a cluster of computers. In this case, work is in the form of web requests, cron jobs, and tasks. So a queue of work is coming in. That work has to be allocated to resources in a reasonable way. So what handles work? Instances. Instances take memory, CPU, disk and bandwidth to run, which means they must be paid for. And because they take a fixed time to spin up, simply spinning up an instance must be paid for, regardless of much work it performs. And because they take a fixed cost to run, it makes sense to leave them around for a while in case there's more load. GAE does not know the load you'll be generating. Load to GAE is generated by a stochastic process.

This is an incredibly hard problem to solve, if it's solvable at all. In real-time scheduling, which this is similar to, it's not possible to deterministically schedule a variable workload. Running the same workload through the system will give different results every time. That' a strange consequence of the change from CPU pricing to instance pricing.

A lot of people have complained that the GAE scheduler is an inherent conflict of interest. The scheduler is determining how many instances will run which in turn determines profitability. Does that seem like a good idea? The incentives are wrong. It would be better if developers and GAE both had a similar incentives, now they are inherently at odds.

Also, notice how Google has this same edge in advertising. Google indexes the data, creates the market, takes the advertising, sets the prices, and determines when advertising will be shown. It's all opaque to the masses as is GAE's scheduler. I'm reminded of the Medici's of Florence who controlled the outcome of public elections by simply controlling who could run.

An interesting area of potential conflict is in oversubscription. If you give a lot of RAM to each application then Google can't run very many instances on the same machine, so RAM has been limited to 128MB on front-ends and 1GB on back-ends. Like any other hosting service the temptation is to oversubscribe these resources to make more money per machine. That will cause swapping, thrashing, etc which will cause higher latencies which will cause more instances to be allocated. That's money in both direction. And the low memory bounds make it harder to write larger multi-threaded applications which are one of the strategies for reducing for reducing latency and decreasing the number of instances. A strange set of bed fellows.

A lot of pressure is being put on the scheduler to fix all problems. We'll just make the schedule better and that will contain your costs. Personally, I'm not sure that will really work because of the granularity mismatch problem we discussed earlier.

The problem is the programmer is in charge of balancing all these objectives. Gubbi expresses the sense of this double bind:

> *My gripe is, the new pricing brings latency into focus, while the developers have nothing but their app code to optimize it. The responsibility for latency is both on the application as well as underlying infrastructure.*

Instead of a policy driven architecture that would tradeoff implementations based on goals, the programmer is having to hardcode a model of programming into their program. When those assumptions change it's hard to change the code. This all needs to be moved up an abstraction level. For example, when a task is low priority then the scheduler could say hay, we don't need to spin up instances for that, let's just schedule it whenever to take advantage of idle time.

Steve has made a good series of scheduler change suggestions in Scheduler profiles and request profiling, the main idea being the ability to *filter requests into predefined scheduling profiles based on the request headers and any other information we can get at that level.*

The Architecture Shift

From Barry Hunter:

> *The 'old' way, pretty much promoted low cpu use, even if that came at the expense of latency. The slow requests, would necessitate lots of instances - costing Google.*
>
> *The 'new' way promotes keeping your latency down. Quick requests gives higher queries per second (per instance). Meaning less instances.*
>
> *By charging for actual instance usage, they are promoting keeping the number of instances down.*
>
> *So the developer should aim to keep the number of instances down - saving money.*
>
> *People who need, want, (or can't be bothered to*

optimise to avoid), slow requests will pay - closer to what it actually costs Google.

Google doesn't want you spinning up instances and tearing them down quickly. Its that spinning up, that 'costs'.

Trouble Shooting Guide

johnP has a good idea of starting a trouble shooting guide for dealing with the price changes. His first draft is:

Excessive instances cost?

- sudden parallelism
- idle instance setting
- decrease response time
- other?

Excessive writes:

- decrease unneeded indexes

Excessive Reads:

- make sure your fetch() rather than looping through results
- check offset queries
- other?

Checkout Google's Documentation

Google has some good doc worth looking at:

- Managing Your App's Resource Usage
- Adjusting Application Performance
- Introduction to Instances

Optimize for One Instance

Joshua Smith:

> *I'm optimizing for the one-instance case, I actually just implemented a trivial cache. The first hit from the kiosk generates the page and returns it, and saves the page in memory. The second hit gets the saved page from memory. It's actually better than my original solution because I'm protected from generating any exceptions when the kiosk does the page reload.*

Task Scheduler Bunching

Joshua Smith:

> *You guys should look at the way task queues interact with the scheduler, though. It appears that the tendency of task queues to bunch up tasks and fire them in parallel leads to a tendency for the scheduler to spin up new instances. I'd suggest that you consider the number of pending tasks in the queue before spinning up an instance just to handle the queue. If there are only a couple ready to go, then just make them wait a little longer. (Of course, giving the users control of this would be best, by having a "priority" setting per-task.)*

Dump Writes to Queue

peterk:

The first type of request, the write will be put on a task queue and it'll return immediately. The queue will be handled by a backend instance. Now this costs a bit upfront, but the key advantage is that I basically get a lot of control over the rate at which my 'write queue' is processed. AppEngine won't spin up new front end instances to handle this for me. I get control over how fast and at how much cost my writes will be handled. There may be more of a lag before writes manifest in the datastore, but I think I can tolerate that in my app.

Use Cursors

Tim Hoffman:

Have a look at cursors too, they don't suffer the performance problem of limit/offsets if you want to work gradually through a data set.

Use Pull Queues, Backends, and Cron

Tim Hoffman:

I would also consider using a pull queue for your task workload rather than normal push queues, that way you could use a cron to limit the number of tasks spun, up, and work through your offline processing in a more sequential fashion. The more I think about it, the more I think your processing approach probably suits backends and cron rather than normal frontend and task queues.

Two Stage Gets Preferred

Google App Engine Post-Preview Pricing FAQ:
Under the new scheme it is still more economical to do a keys-only query that fetches 1000 keys, and then does a get on the 500 of them compared to running a regular (non keys-only) query for all 1000 directly. The first is option more economical. Fetching 1000 keys + fetching 500 entities = $0.0001 + 0.00035 = $0.00045; fetching 1000 entities = $0.0007.

Stop the Bots

In Googlebot Spawning New Instances, Jeff Deskins notices something very interesting:

> *Thought it was funny that while I was looking at ways to keep the cost down in App Engine - Googlebot started crawling the site at a rate of up to 12 pages per second. This caused another 6 instances to be created!*

Another example of low priority spiky behaviour driving up costs. With each crawl costing money, the whole idea of crawling the Internet will have to change.

Spiky App Penalty

Jan Z/ Hapara:

> *This change penalizes "spiky" apps in a huge way. Three years ago, when we first started with GAE, we did so for three reasons: 1) scalability 2) pay for what you use model, and 3) SIMPLICITY.*
>
> *Our app is very "spiky" - a single user will at times*

generate 50+ concurrent requests that need to be served quickly. We've spent three years working around the GAE to fine-tune the code and work within the scalability and other constraints, and it works remarkably well. There is not a chance that we could get as far if we had to use EC2 or other systems.

The new pricing model has two implications for us: a) we will start paying mostly for idle instances and b) we lose the simplicity and predictability of the existing model. It seems that you're shifting the "hard" problem here back to the users, and that's a shame.

As others have suggested, the new scheduler needs to have the flexibility to cater to small and tight apps - in not providing it you are penalizing the very people who embraced your tools first and worked hard around your early limitations and teething problems.

Cost Uncertainty

The scheduler to a programmer is a black box hiding a highly dimensional algorithm that is little understood. It can't be controlled. It can't be reasoned with. Controlling the number of instances and bandwidth will be a lot harder than controlling how much time a CPU application uses. This leads to uncertainty confusion. Is this just part of the new world order?

Instance Idea Includes both CPU and RAM

FAQ Author Greg D on why GAE didn't institute a separate charge for RAM:

We have in essence added a RAM charge by charging *for Instances. By having an Instance up with an allocated amount of memory you are essentially using that RAM. So, by charging for the Instance we are charging you for the combination of the RAM and CPU. We considered splitting this charge out so we would continue to charge CPU-hours and then also charge Instance-hours (which we could've called RAM-hours). This both seemed more confusing as well as would not have been cheaper, so it didn't seem worthwhile. I know that there has been a lot of discussion about charging this way instead. In the end it, whether you call it RAM-hours or Instance-hours and whether or not you charge for CPU-hours on top of it, it would end up with the same result. Which is that applications are charged for the amount of RAM allocated and the amount of time it is allocated for. This means applications that want to save money will need to optimize around using fewer RAM-hours which in essence means taking less time to get things done.*

Multitasking is Back Baby

With instance based pricing the goal is to get as high a utilization out of an instance as possible. This requires multithreading so as many simultaneous operations can be executing in a container as possible. Previously GAE was single threaded. More instances were started to handle more requests. Obviously not cost efficient, so we are back to writing threaded code again. Which is fine really. Starting a huge JVM is exactly why we went to application servers in the first place and away from a CGI model. That GAE's model wasn't profitable is just a little disappointing.

The gains from multi-threading can be so high GAE is counting on them balancing the cost increases. Brandon Wirtz :

> *I just finished a re-write of an early version of my App from Python to Java. This is not my full Latest Release app, but it is the same as a version that has been deployed for months. Both versions do the same things, are compatible in every way. I haven't been running that long, but it appears I have* ***gone from 20 instances down to 1****. I'll give a full report in 8 hours when I have enough data to know for sure.*
>
> *I fully expect that the scale factor of this app will be a strong indicator of the scale of the full app. (This re-write took about 3 hours and is almost a line for line translation)*

Memory consumption goes up with concurrency and the multithreading, so it will be interesting to see how these work (or don't) together.

Is GAE is Still a Good Deal Compared to Amazon?

From Greg D:

> *You are comparing very different services on the basis of RAM alone. RAM is one of many factors which dictate what you can accomplish with a service. App Engine also includes a slew of free APIs, no need to manage your App (the newly exposed Scheduler knobs are a convenience rather than a necessity), no need to run and maintain an OS (which typically consumes*

quite a bit of RAM), etc. App Engine is a Platform, when it comes down to it we've decided to use resources (both electronic and human) differently and so we are charging differently for it as well.

Robert Kluin:

*When comparing prices, don't underestimate the value of the HR datastore, scalable front-ends, memcache, *and* the management costs of those things. I've managed geographically distributed databases, if you've not done it -- it is significantly more involved than you think (assuming you care about your data). Don't forget to factor in the cost of that management into your comparisons; if things run very smoothly it will still be several thousand dollars per year.*

Barry Hunter:

For what it worth, I dont think it fair to compare a GAE instance like for like to a VPS.

they are very different 'beasts'. GAE in theory provides you much more. Google handle all the setup, configuration, provisioning, load balancing.

Granted it does seem expensive, if you only using equivalent of one VPS. But once you start need to manage multiple VPSs, provision load-balancers, re-provision in case of failure etc, make backups. All that happens 'magicically' with GAE.

Don't forget you also get a free quota of instance hours

> *too.*

The Backend is Expensive

Ugorji:

> *Backend Instances seem ridiculously and prohibitively high ($115/month for a resident 256MB/1.2GHz instance). Their prohibitive costs makes them very unattractive for a lot of users who could leverage this functionality, causing us to look elsewhere.*
>
> *In a way, it seems we're caught between a rock and a hard place. Backends are an excellent way to do long-running CPU-intensive actions, allowing us move away from the current practice of spawning chains of tasks which each complete in a set time, or using a map-reduce operation. However, the backends are so expensive that most people will not use them. Unfortunately, the current practice of using frontend instances and tasks/map-reduce is now expensive also because for each instance, we have to pay an extra 1/4 instance hour tax beyond our usage.*

Performance = Scalability

FAQ for out of preview pricing changes:

> *Therefore there is a direct relationship between the latency and number of requests which can be handled on the instance per second, for instance: 10ms latency = 100 request/second/Instance, 100ms latency = 10*

request/second/Instance, etc. Multi-Threaded Instances can handle many concurrent requests. Therefore there is a direct relationship between the cpu consumed and the number of requests/second. For instance, for a B4 (approx 2.4GHz) instance: consuming 10 Mcycles/request = 240 request/second/ Instance, 100 Mcycles/request = 24 request/second/ Instance, etc. These numbers are the ideal case but they are pretty close to what you should be able to accomplish on an Instance. Multi-Threaded instances are currently only supported for Java; we are planning support for Python later this year.

Jeff Schnitzer:

*High-latency requests don't scale *on appengine* because of specific design decisions made by Google. High-latency requests scale just fine on other platforms - the folks running Node.js/Tornado/EventMachine/etc have no problem whatsoever with tens of thousands of latent requests. Even traditional java appservers handle latency with acceptable limits - depending on what you're doing, you can get away with hundreds or even thousands of concurrent requests per JBoss instance.*

No More Ad Supported Websites on GAE

Kaan Soral:

Let's say I have an application that depends on ad income. And assume every request is 1 second and Task Scheduler is

> *perfect! So I pay $0.05 to an instance every hour, which has 3600 seconds inside it. So lets average that number to 3600 requests and lets say there are 1800 page views. (Assuming things are done with Ajax). So the cost for 1000 page views are: $0.05/1800*1000=0.027 $. Assuming everything works perfect, and not counting background tasks, although mine are huge, I need to get at least 0.027$ ecpm. For example for Turkish traffic sometimes ecpms can drop below 0.1 $'s, and I am sure there are countries out there with significantly lower ecpms and our cost traffics were optimal. To sum up, It seems if I use GAE, I will always have the risk that the costs will be higher than the income ...*
> *I have applications on Dedicated Servers, usually a server is nearly idle, the load is 0.5 out of 8, sometimes 2-3, and I am sure I don't utilize them to %20 maybe, but still my cost ratio is %10 ! If I could utilize a server to a maximum level that could be %2 ! And on the best case it seems this rate will be %25 on gae assuming everything is perfect ....*

The implication for mobile is that using GAE as a cheap backend for mobile clients won't be viable either.

Hybrid Architectures?

Tim Meadowcroft:

> *Or integration across clouds - I anticipate more people taking the user account/authentication of GAE, putting their static files on one or more CDNs, using 3rd party*

> *pub-sub services for updates to clients, perhaps federating data across different services (hot data vs reference data), keeping fixed dedicated instances for background processes that don't suffer traffic surges etc. I'd expect to see abstraction services - libraries that let you write to one API for a type of service and provide independence from any specific underlying platform.*

Python or Java?

Java has JIT, so assuming instances will stay up long enough for just-in-time compiling to kick in, and given the penalty for higher latencies, will Java become preferred over Python on GAE?

The whole Instances Pricing thing is Too Complicated

Daniel Florey with a great observation on how complex this all is:

> *Is there no way to create a simple pricing model that will not force me to learn everything about the internals of the app engine scheduler? I'm fine with paying for consumed resources at a reasonable price but I'm not feeling comfortable with the instances/hours approach. It makes me feel like the scheduler impl is responsible for my bill. I'd prefer a simple approach (from a users perspective) where I would get charged for the cpu resources/memory consumed + be able to spend a budget on "scalability/processing power" and let app engine take care of whatever it needs to speed up / slow down my app according to my settings.*

Conclusion

- **It's great GAE survived the axe.** Hopefully they've put themselves on a road to sustainability. It took a lot of courage to make such a radical change. It must have been gut wrenching. The good news is we'll get to see the GAE team still out there inventing and innovating. The bad news is that we lost one of the greatest innovations of the last few years, the CPU based billing model. Will we see it again? Is that dream really dead or just waiting to rise again?
- **You shall not PaaS**. Part of the promise of PaaS is programming simplicity for developers. In real-life, at scale, is that possible? For performance, in GAE, programmers are responsible for building a caching layer into their software, in the same way traditional web architectures have been doing for years. Ensuring cache consistency is a huge burden. We've also seen how queue games have to be played to stop instances from spinning up unnecessarily. Programming is not zero maintenance using the scheduler. Developer time is valuable. That is a strong argument for using a zero maintenance PaaS system. But if programmer time must continually be spent optimizing for the scheduler, is PaaS still a better investment?
- **Half way there**. In Cloud Programming Directly Feeds Cost Allocation Back Into Software Design and Building Super Scalable Systems: Blade Runner Meets Autonomic Computing In The Ambient Cloud we talked about how software can change in response to the cost gradient of the underlying platform. These type of discussions have been going strong on how best to use GAE's new scheduler. It's clear we need to operate at a higher level of abstraction to make this model practical. Sure, we can parametrize the model by adding more knobs, but that quickly overwhelms programmers with options that have unknowable

interactions and consequences. A new synthesis must be made.

- **The program-machine mismatch**. Instance based billing is required because programs and machine resources are packaged in different sized quanta. Virtualization has helped smoothed out this mismatch, but it is really it's just the same as we ever had on more capable machines. A different architecture will be necessary to fully exploit hardware resources in such a way that programmers can get the usage based billing they so desire.
- **Huge demand for a scalable, low cost application infrastructure**. Face it, most applications just scrape by. They are ad supported, charge a whopping 99 cents on the app store, or have such low conversion rates that 1% pay for the other 99%. Not everyone is Apple, Zynga, or Netflix. A lot of ideas are just cool enough to survive in a supportive environment. That's not what we have today. What are the great masses on the margins supposed to do for infrastructure? Charge more? Sure. Innovation is being held back because there's a lack of affordable infrastructure at scale that can be made profitable at a low margin. It looks like GAE won't solve that problem. But somebody must.

Related Articles

- The Three Ages Of Google - Batch, Warehouse, Instant
- GAE SLA
- Adjusting Application Performance
- Google App Engine Price Increases - Bad for Small Apps, and No More Free Scaling by Danny Tuppeny
- App Engine pricing changes revisited by Stavros Korokithakis
- FAQ Requests for Guido and Greg on GAE Python’s “Massive Concurrency”
- Is MapReduce still a flexible solution on AppEngine under the new pricing model?

- FAQ for out of preview pricing changes
- New App Engine Pricing policy, the good the bad and the ugly by σ χ • λ ι α
- The Year Ahead for Google App Engine
- AppEngine 1.5.0 Release
- On Hacker News: The Year Ahead for Google App Engine, Goodbye, Google App Engine, Google App Engine 1.5.0 released
- Vannie Totaro put together a very good list of new GAE links.
- Is App Engine suddenly becoming more expensive???
- On Reddit: RIP AppEngine, Rip AppEngine
- The Unofficial Guide to Migrating Off of Google App Engine by Eric Silverberg
- GAE Pricing Changes - Sucker Punching the Development Community
- The price of Scalability
- Story of an Android Developer
- Google App Engine's new pricing model by Guillaume Laforge
- Google faces backlash over new App Engine pricing by Srirangan
- Keep it short: Who is forced to leave GAE?
- Managing Your App's Resource Usage
- Introduction to Instances
- The Amazing Story Of AppEngine And The Two Orders Of Magnitude by Emlyn O'Regan
- Really? We're supposed to throttle our visitors?
- Why app engine new price model is totally wrong
- Google App Engine Pricing Angers Developers, Kills PlusFeed
- How I reduced my AppEngine costs despite new pricing
- Coders howl over Google's App Engine price hike (natch) by Cade Metz

- Frontend instance hours seem to be a bit overcharged
- New Prices + Really Small Apps - what will happen?
- Why I think it's the time to leave GAE
- The scheduler needs a fix and quick
- anybody else do a bill comparison
- App Engine is finished, here's why
- UNBELIEVABLE NEW BILLING!! Google Please respond...
- New pricing will cost me over 500 $ monthly for my three sites... I am floored.
- Why I will Likely Be Leaving GAE (30x increase in Price)
- I really like the new pricing...
- Looking for ideas on eliminating instance use
