Stack Overflow Architecture

Wednesday, August 5, 2009 at 9:36AM

Todd Hoff in Example, Microsoft

Update 2: Stack Overflow Architecture Update - Now At 95 Million Page Views A Month
Update: Startup – ASP.NET MVC, Cloud Scale & Deployment shows an interesting alternative approach for a Windows stack using ServerPath/ GoGrid for a dedicated database machine, elastic VMs for the front end, and a free load balancer.

Stack Overflow is a much loved programmer question and answer site written by two guys nobody has ever heard of before. Well, not exactly. The site was created by top programmer and blog stars Jeff Atwood and Joel Spolsky. In that sense Stack Overflow is like a celebrity owned restaurant, only it should be around for a while. Joel estimates 1/3 of all the programmers in the world have used the site so they must be serving up something good.

I fell in deep like with Stack Overflow for purely selfish reasons, it helped me solve a few difficult problems that were jabbing my eyes out with pain. I also appreciate their no-apologies anthropologically based design philosophy. Use design to engineer in the behaviours you want to encourage and minimize the responses you want to discourage. It's the conscious awareness of the mechanisms that creates such a satisfying synergy.

What is key about the Stack Overflow story for me is the strong case they make for scale up as a viable solution for a certain potentially large class of problems. The publicity these days is all going scale out using NoSQL

databases.

If you need to Google scale then you really have no choice but to go the NoSQL direction. But Stack Overflow is not Google and neither are most sites. When thinking about your design options keep Stack Overflow in mind. In this era of multi-core, large RAM machines and advances in parallel programming techniques, scale up is still a viable strategy and shouldn't be tossed aside just because it's not cool anymore. Maybe someday we'll have the best of both worlds, but for now there's a big painful choice to be made and that choice decides your fate.

Joel boasts that for 1/10 the hardware they have performance comparable to similarly size sites. He wonders if these other sites have good programmers. Let's see how they did it and you be the judge.

Site: http://stackoverflow.com

The Stats

- 16 million page views a month
- 3 million unique visitors a month (Facebook reaches 77 million unique visitors a month)
- 6 million visits a month
- 86% of traffic comes from Google
- 9 million active programmers in the world and 30% have used Stack Overflow.
- Cheaper licensing was attained through Microsoft's BizSpark program. My impression is they pay about $11K for OS and SQL licensing.
- Monitization strategy: unobtrusive adds, job placement ads, DevDays conferences, extend the software to target other related niches (Server Fault, Super User), develop StackExchange as a white label and self hosted version of Stack Overflow, and perhaps develop some sort of programmer rating system.

Platform

- Microsoft ASP.NET MVC
- SQL Server 2008
- C#
- Visual Studio 2008 Team Suite
- JQuery
- LINQ to SQL
- Subversion
- Beyond Compare 3
- VisualSVN 1.5
- Web Tier

- 2 x Lenovo ThinkServer RS110 1U
- 4 cores, 2.83 Ghz, 12 MB L2 cache
- 500 GB datacenter hard drives, mirrored
- 8 GB RAM
- 500 GB RAID 1 mirror array

- Database Tier

- 1 x Lenovo ThinkServer RD120 2U
- 8 cores, 2.5 Ghz, 24 MB L2 cache
- 48 GB RAM

- A fourth server was added to run superuser.com. All together the servers also run Stack Overflow, Server Fault, and Super User.
- QNAP TS-409U NAS for backups. Decided not to use a cloud solution because the bandwidth costs of transferring 5 GB of data per day becomes prohibitive.
- Hosting at http://www.peakinternet.com/. Impressed with their detailed technical responses and reasonable hosting rates.
- SQL Server's full text search is used extensively for the site search and detecting if a question has already been asked. Lucene.net is considered an attractive alternative.

Lessons Learned

This is a mix of lessons taken from Jeff and Joel and comments from their posts.

- If you're comfortable managing servers then buy them. The two biggest problems with renting costs were: 1) the insane cost of memory and disk upgrades 2) the fact that they [hosting providers] really couldn't manage anything.
- Make larger one time up front investments to avoid recurring monthly costs which are more expensive in the long term.
- Update all network drivers. Performance went from 2x slower to 2x faster.
- Upgrading to 48GB RAM required upgrading MS Enterprise edition.
- Memory is incredibly cheap. Max it out for almost free performance. At Dell, for example, upgrading from 4G memory to 128G is $4378.
- Stack Overflow copied a key part of the Wikipedia database design. This turned out to be a mistake which will need massive and painful database refactoring to fix. The refactorings will be to avoid excessive joins in a lot of key queries. This is the key lesson from giant multi-terabyte table schemas (like Google's BigTable) which are completely join-free. This is significant because Stack Overflow's database is almost completely in RAM and the joins still exact too high a cost.
- CPU speed is surprisingly important to the database server. Going from 1.86 GHz, to 2.5 GHz, to 3.5 GHz CPUs causes an almost linear improvement in typical query times. The exception is queries which don't fit in memory.
- When renting hardware nobody pays list price for RAM upgrades unless you are on a month-to-month contract.
- The bottleneck is the database 90% of the time.
- At low server volume, the key cost driver is not rackspace, power, bandwidth, servers, or software; it is NETWORKING EQUIPMENT. You

need a gigabit network between your DB and Web tiers. Between the cloud and your web server, you need firewall, routing, and VPN devices. The moment you add a second web server, you also need a load balancing appliance. The upfront cost of these devices can easily be 2x the cost of a handful of servers.

- EC2 is for scaling horizontally, that is you can split up your work across many machines (a good idea if you want to be able to scale). It makes even more sense if you need to be able to scale on demand (add and remove machines as load increases / decreases).
- Scaling out is only frictionless when you use open source software. Otherwise scaling up means paying less for licenses and a lot more for hardware, while scaling out means paying less for the hardware, and a whole lot more for licenses.
- RAID-10 is awesome in a heavy read/write database workload.
- Separate application and database duties so each can scale independently of the other. Databases scale up and the applications scale out.
- Applications should keep state in the database so they scale horizontally by adding more servers.
- The problem with a scale up strategy is a lack of redundancy. A cluster ads more reliability, but is very expensive when the individual machines are expensive.
- Few applications can scale linearly with the number of processors. Locks will be taken which serializes processing and ends up reducing the effectiveness of your Big Iron.
- With larger form factors like 7U power and cooling become critical issues. Using something between 1U and 7U might be easier to make work in your data center.
- As you add more and more database servers the SQL Server license costs can be outrageous. So by starting scale up and gradually going scale out with non-open source software you can be in a world of financial hurt.

It's true there's not much about their architecture here. We know about their machines, their tool chain, and that they use a two-tier architecture where they access the database directly from the web server code. We don't know how they implement tags, etc. If interested you'll be able to glean some of this information from an explanation of their schema.

Discussion

As an architecture profile candidate Stack Overflow has earned two important HighScalability badges: the Microsoft Stack Badge and and the Scale Up Badge. Both controversial and interesting topics of discussion.

Microsoft Stack Badge

The Microsoft Stack Badge was earned because Stack Overflow uses the entire Microsoft Stack: OS, database, C#, Visual Studio, and ASP .NET. People are always interested in how MS compares to LAMP, but I don't have many case studies to show them.

Markus Frind of Plenty of Fish fame is often used as a Microsoft stack poster child, but since he explicitly uses as little of the stack as possible he's not really a good example. Stack Overflow on the other hand is brash in proclaiming their love for MS, even when that love is occasionally spurned.

It's hard to separate out the Microsoft stack and the scale up approach because for licensing reasons they tend to go together. If you find yourself in the position of transitioning from scale up to scale out by adding dozens of cores, MS licensing will bite you.

Licensing aside I personally find C#, Visual Studio, and .Net a very productive environment. C#/.Net is at least as good as Java/JVM. ASP .NET has always been a confusing mess to me. The knock against SQL

Server is you have to pay for it and if that doesn't bother you then it's a solid choice. The Windows OS may not be as solid as other alternatives but it works well enough.

So for a scale up solution a Microsoft stack works, especially if you are already Windows centric.

Scale Up Badge

This won't be a reenactment of the scale out vs scale up vs rent vs buy wars. For a thorough discussion of these issues please take a look at Scaling Up vs. Scaling Out and Server Hosting — Rent vs. Buy?. If you aren't confused and if your head doesn't hurt after reading all that then you haven't properly understood the material :-)

The Scale Up Badge was awarded because Stack Overflow uses a scale up strategy to meet their scaling requirements. When they reach a limit they scale vertically by buying a bigger machine and adding more memory.

Stack Overflow is in the sweet spot for scale up. It's not too large, but with an Alexa ranking of 1,666 and 16 million page views a month it's still a substantial site. Not Google scale, and probably will never have to be, but those are numbers many sites would be thrilled to have. Yet they aren't uploading large amounts of media. They aren't dealing with billions of tweets across complex social networks with millions of users. Their number of users is self limiting. And there are still directions they can take if they need to scale (caching, more web servers, faster disks, more denormalization, more memory, some partitioning, etc). All-in-all it's a well done and very useful two-tier CRUD application.

NoSQL is Hard

So should Stack Overflow have scaled out instead of up, just in case?

What some don't realize is NoSQL is hard. Relational databases have many many faults, but they make a lot of common tasks simple while hiding both the cost and complexity. If you want to know how many black Prius cars are in inventory, for example, then that's pretty easy to do.

Not so with most NoSQL databases (I'll speak generally here, some NoSQL databases have more features than others). You would have program a counter of black Prius cars yourself, up front, in code. There are no aggregate operators. You must maintain secondary indexes. There's no searching. There are no distributed queries across partitions. There's no Group By or Order By. There are no cursors for easy paging through result sets. Returning even 100 large records at time may timeout. There may be quotas that are very restrictive because they must limit the amount of IO for any one operation. Query languages may lack expressive power.

The biggest problem of all is that transactions can not span arbitrary boundaries. There are no ACID guarantees beyond a single record or small entity group. Once you wrap your head around what this means for the programmer it's not a pleasant prospect at all. References must be manually maintained. Relationships must be manually maintained. There are no cascading deletes that act correctly during a failure. Every copy of denormalized data must be manually tracked and updated taking into account the possibility of partial failures and externally visible inconsistency.

All this functionality must be written manually by you in your code. While flexibility to write your own code is great in an OLAP/map-reduce situation, declarative approaches still cover a lot of ground and make for much less brittle code.

What you gain is the ability to write huge quantities of data. What you lose is complacency. The programmer must be very aware at all times that they are dealing with a system where it costs a lot to perform distribute

operations and failure can occur at anytime.

All this may be the price of building a truly scalable and distributed system, but is this really the price you want to pay?

The Multitenancy Problem

With StackExchange Stack Overflow has gone into the multi-tenancy business. They are offering StackExchange either self-hosted or as a hosted white label application.

It will be interesting to see if their architecture can scale to handle a large number of sites. Salesorce is the king of multitenancy and although it's true they use Oracle as their database, they basically use very little of Oracle and have written their own table structure, indexing and query processor on top of Oracle. All in order to support multitenancy.

Salesforce went extreme because supporting a lot of different customers is way more difficult than it seems, especially once you allow customization and support versioning.

Clearly all customers can't run in one server for security, customization, and scaling reasons.

You may think just create a database for each customer, share a server for a certain number of customers, and then add more servers as needed. As long as a customer doesn't need more than one server you are golden.

This doesn't seem to work well in practice. Oddly database managers aren't optimized for adding or updating databases. Creating databases is a heavyweight operation and can degrade performance for existing customers as system locks are taken. Upgrade issues are also problematic. Adding columns locks tables which causes problems in high traffic situations. Adding new indexes can also take a very long time and degrade

performance. Plus each customer will likely have specializations that makes upgrading even more complicated.

To get around these problems Salesforce's Craig Weissman, Chief Architect, created an innovative approach where tables are not created for each customer. All data from all customers is mapped into the same data table, including indexes. The schema for that table looks something like orgid, oid, value0, value1...value500. "orgid" is the organization ID and is how data is never mixed up. It's a very wide and sparse table, which Oracle seems to handle well. Hundreds and hundreds of "tables" and custom fields are mapped into the data table.

With this approach Salesforce has no option other than to build their own infrastructure to interpret what's in that table. Oracle is left to handle transactions, concurrency, and deadlock detection. The advatange is because there's an interpreted layer handling versions and upgrades is relatively simple because the handling logic can be baked in. Strange but true.

Related Articles

This list includes a number of posts by Jeff as he chronicles their journey with Stack Overflow. Jeff is wonderful about being open about what they are doing and why. The comment threads are often tremendous. There's a lot to learn.

- Learning from StackOverflow.com by Joel Spolsky
- Scaling Up vs. Scaling Out: Hidden Costs by Jeff Atwood
- What Was Stack Overflow Built With?
- New Stack Overflow Server Glamour Shots
- New Stack Overflow Servers Ready
- Server Hosting — Rent vs. Buy? - this is a very informative discussion the pros and cons of renting vs buying.

- Rent vs. Buy (or EC2 vs. building your own iron) by Michael Friis
- Oh, You Wanted "Awesome" Edition - We recently upgraded our database server to 48 GB of memory -- because hardware is cheap, and programmers are expensive.
- Our Backup Strategy - Inexpensive NAS
- The Economics of Bandwidth
- Understanding the StackOverflow Database Schema by Brent Ozar
- Server Speed Tests - new hardware 2x slower - it was the network.
- ASP.NET MVC: A New Framework for Building Web Applications
- Three key things to know about moving MySQL into the cloud by morgan
- NoSQL Conference
- Decline of the Enterprise Data Warehouse by Bradford Stephens
- Webinar: Multitenant Magic - Under the Covers of the Force.com Data Architecture by Craig Weissman, Chief Architect, salesforce.com.
