Why are Facebook, Digg, and Twitter so hard to scale?

Tuesday, October 13, 2009 at 7:35AM

Todd Hoff in Example, Strategy, facebook

Real-time social graphs (connectivity between people, places, and things). That's why scaling Facebook is hard says Jeff Rothschild, Vice President of Technology at Facebook. Social networking sites like Facebook, Digg, and Twitter are simply harder than traditional websites to scale. Why is that? Why would social networking sites be any more difficult to scale than traditional web sites? Let's find out.

Traditional websites are easier to scale than social networking sites for two reasons:

- They usually access only their own data and common cached data.
- Only 1-2% of users are active on the site at one time.

Imagine a huge site like Yahoo. When you come to Yahoo they can get your profile record with one get and that's enough to build your view of the website for you. It's relatively straightforward to scale systems based around single records using distributed hashing schemes. And since only a few percent of the people are on the site at once it takes comparatively little RAM cache to handle all the active users.

Now think what happens on Facebook. Let's say you have 200 friends. When you hit your Facebook account it has to go **gather the status of all 200 of your friends at the same time** so you can see what's new for them. That means 200 requests need to go out simultaneously, the replies need to be merged together, other services need to be contacted to get more details, and all this needs to be munged together and sent through PHP and a web server so you see your Facebook page in a reasonable amount of time. Oh my.

There are several implications here, especially given that on social networking sites a high percentage of users are on the system at one time (that's the social part, people hang around):

- All data is active all the time.
- It's hard to partition this sort of system because everyone is connected.
- Everything must be kept in RAM cache so that the data can be accessed as fast as possible.

Partitioning means you would like to find some way to cluster commonly accessed data together so it can be accessed more efficiently. Facebook, because of the interconnectedness of the data, didn't find any clustering scheme that worked in practice. So instead of partitioning and denormalizing data Facebook **keeps data normalized and randomly distributes** data amongst thousands of databases.

This approach requires a very fast cache. Facebook uses memcached as their caching layer. All data is kept in cache and they've made a lot of modifications to memcached to speed it up and to help it handle more requests (all contributed back to the community).

Their **caching tier services 120 million queries every second** and it's the core of the site. The problem is memcached is hard to use because it requires programmer cooperation. It's also easy to corrupt. They've developed a complicated system to keep data in the caching tier consistent with the database, even across multiple distributed data centers. Remember, they are caching user data here, not HTML pages or page fragments. Given how much their data changes it's would be hard to make page caching work.

We see similar problems at Digg. Digg, for example, must deal with the problem of sending out updates to 40,000 followers every time Kevin Rose diggs a link. Digg and I think Twitter too have taken a different approach than Facebook.

Facebook takes a **Pull on Demand** approach. To recreate a page or a display fragment they run the complete query. To find out if one of your friends has added a new favorite band Facebook actually queries all your friends to find what's new. They can get away with this but because of their awesome infrastructure.

But if you've ever wondered **why Facebook has a 5,000 user limit** on the number of friends, this is why. At a certain point it's hard to make Pull on Demand scale.

Another approach to find out what's new is the **Push on Change** model. In this model when a user makes a change it is pushed out to all the relevant users and the changes (in some form) are stored with each user. So when a user want to view their updates all they need to access is their own account data. There's no need to poll all their friends for changes.

With security and permissions it can be surprisingly complicated to figure out who should see an update. And if a user has 2 million followers it can be surprisingly slow as well. There's also an issue of duplication. A lot of duplicate data (or references) is being stored, so this is a denormalized approach which can make for some consistency problems. Should permission be consulted when data is produced or consumed, for example? Or what if the data is deleted after it has already been copied around?

While all these consistency and duplications problems are interesting, Push on Change seems the more scalable approach for really large numbers of followers. It does take a lot of work to push all the changes around, but that can be handled by a job queuing system so the work is distributed across a cluster.

The challenges will only grow as we get more and more people, more and deeper inter-connectivity, faster and faster change, and a greater desire to consume it all in real-time. We are a long way from being able to handle this brave new world.
