Making Internet Services Real

April 27, 2005

I. Issues beyond the Prototype
   - Scalability
   - Availability
   - Rapid Growth
   - Evolution
   - Personalization
   - Logging

II. Scalability

Must handle 10M hits/day

General approach: replicate simple boxes for more throughput

Problem: data must be completely replicated and then kept in sync
   - how do you update all nodes simultaneously?
   - do the nodes have to be the same?

Problem: how do you load balance?
   - DNS round robin: assign several IP addresses to the same domain name
   - Level 4 switch: redirect port 80 traffic per connection (this is hard, but you can buy switches that do it)
   - Router changes: WCCP protocol for caches

Problem: what if too much data to replicate
   - example: search engine database?
   - solution: (Inktomi) spread the database across the nodes and use parallel computing techniques for each query

III. High Availability

   - goal: keep handling queries all the time
   - non-goal: make each query complete correctly! (that is fault tolerance, which is much harder). HA is good enough if we assume that users hit reload, and that the probability of node failure is low
   - tricky part: how to keep new queries from going to the down node. DNS round robin does not help, because DNS time-to-live is too long. Level 4 switches can do this.
Web Scaling

“virtual IP address stealing” is the process of live nodes stealing the (multiple) IP addresses of a down node, and handling their future queries. Problem if only one IP/node: the stealing node gets double the traffic, which it probably can’t handle...

- fast restart is very important in practice
- avoiding failure is also important, but harder to deliver for changing software
- maintenance is tricky: need to gracefully take nodes out of service. This must be integrated with the load balancing and HA techniques, but is usually easier than dealing with faults, as the “failure” is predictable and thus can be planned for...

IV. IV. Rapid Growth

Problem: must respond quickly to growth in traffic

1) Can we predict growth? not very well yet… There are seasonal effects (weeks after Christmas have had strong growth) and advertising does seem to work (so have extra capacity in place before the campaign hits).

2) How quickly can we react to unexpected growth? At very best, 24 hours. In practice, a few days to a week. Critical issues: order and delivery of new equipment, installation/testing, and sometimes software changes (e.g. Inktomi has to rearrange database when it adds nodes). Non-standard equipment can take a long time -- big SMPs can be 90 days; this means some parts are best kept on hand just in case. New equipment doesn’t always work, so you may have to order extra or order twice.

3) Deploying new bandwidth can take even longer. One common solution is to use a web hosting facility, which typically maintains excess bandwidth for purchase by customers. They also provide the increased rack space, power and AC you need to grow.

Can you grow the system without taking it off-line?

V. Evolution

Problem: need to change the system, but would like the change to be instantaneous (no down time). This is hard.

Most groups can now do “rolling upgrades” or a “fast switchover”.

Rolling upgrade: upgrade one node at a time in a “wave”. Users will get either the new or the old service depending on which node they reach; problems if the nodes interact...

Fast switchover: reboot all of the machines at once and they come up in the new regime. This is often the only option when there are dependencies among the nodes. The downtime can be seconds to minutes, which may be OK if rare and done in off hours. (Are there off hours?)

Often the ease of evolution depends on the change: simple HTML changes may be made in a rolling upgrade, while database changes may require a system-wide restart.
Web Scaling

VI. Personalization/Customization

Goal: customize content for each user. This greatly increases user loyalty and creates “sticky” sites -- those that keep users coming back and thus produce more revenue.

Related problem: generate different versions for different browsers. This problem is easier because there are relatively few browsers that you would explicitly target. However, it is much less impact on users than personalization.

Issues:
- tracking users
- keeping user profile data
- server-side scripting
- privacy

Tracking users: two basic approaches cookies and “fat URLs”. Cookies are hidden tokens that the browser tracks. The first time, a site hands a cookie to the browser; after that the browser hands it back every time it visits the site. Cookies are just strings and can contain roughly anything the site wants, although cookies are typically limited to 256 bytes.

Fat URLs are URLs that have extra user-specific info embedded within them. New users uses a generic URL and get redirected to a fat URL. The site propagates the extra state to links as you surf through the site. Fat URLs are older than cookies and cookies are typically a better solution. However, fat URLs work with all browsers.

Keeping Profile Data

In both cases, site must decide what state to put in the cookie or fat URL. There are two approaches: 1) put all user state in the cookie (or fat URL), and 2) put just a user id in the cookie (or fat URL).

Complete state: user settings are encoded into the string in some fashion. The big advantage is that the site does not have to store profiles, since each user always brings their own. However, there is a limit to how much state will fit in a cookie. Worse, it is difficult to evolve the state representation, since cookies stay around for a long time.

A better solution is to embed just the user id in the cookie. User ids are small and don’t change, so they work well with cookies. The real state is stored in a database, with the user id as the key. Thus every visit requires a database lookup; this is OK in practice because of caching (in the database). There are at least two big advantages: a) it is easy to add new fields and evolve the schema, b) you can analyze the profiles for patterns and learn about your users and their preferences -- it is much harder to do this if the state is in cookies on your users’ hard drives...

Server-Side Scripting

The most common form is Active Server Pages (ASP). This is essentially embedded visual basic commands that are part of the HTML page. The scripts are executed on the server to generate the real page. This is the correct way to do personalization. (Other scripting languages such as TCL or Perl are fine as well.)
Web Scaling

Typically, high up on the page you load the user profile from the ID in the cookie. This sets a bunch of variables that can then affect the rest of the page.

Scripts in the page use these variables to compute the content of the page.

Example: My Yahoo! stock quotes (not done with ASP)

User profile contains the list of stocks as a string (you can easily see this string when you “edit” your profile). The scripts parse the string and then generate one row in an HTML table for each stock symbol.

Typically scripts include: conditional generation of features, database lookups, e-mail interaction, etc. Good example sites that do this: MyExcite, news.com, wsj.com (Wall Street Journal)

Privacy

Privacy is harder than people think. The easy part is not giving out personal information to third parties without users’ permission. Harder is protecting it from hackers: Netcom lost 20,000 credit card numbers to a hacker (Mitnick).

Harder still is leaking info via cookies and URLs. For example, web browsers typically included the URL that “referred” the user the site. This means that the URL to your site will be in the headers to the next site that users visits. Therefore, you can’t have the password embedded in the URL!

Sites should have privacy policies published online. IBM said it will not advertise on sites that do not have acceptable privacy policies clearly stated on the site.