17 Techniques Used to Scale Turntable.fm and Labmeeting to Millions of Users

Monday, September 26, 2011 at 9:00AM

General Chicken in Example

In How to launch in a month and scale to a million users, Joseph Perla, Former VP of Technology and founding team of Turntable.fm, shares techniques he used to build and quickly scale his startups. The post is very well written and a must read. Here are the essentials:

1. **Keep it simple.** Build API's before making the website or mobile apps. Keep interfaces small and single-purpose.
2. **Get it right.** Build in automated tests from the start. Create function tests, module level tests, and full integration tests. Run tests on every commit. No new code written while bugs exist.
3. **Don't hide power.** Use Pebbles to write bug-free Javascript, a library to create complicated AJAX interactions by writing 0 javascript by adding a few extra HTML tags to code.
4. **Use procedure arguments to provide flexibility in an interface.** Pass functions instead of parameters to support complicated scenarios. For example, a filter function return a boolean.
5. **Leave it to the client.** Keep the server simple and move as much functionality as possible to the client.
6. **Continuity.** Keep interfaces stable. Version interfaces from the start.
7. **Keep secrets of the implementation.** Keep service implementations entirely independent to provide maximum flexibility to handle requirement changes, even though it means a slight performance decrease.

8. **Use a good idea again instead of generalizing it.** It's OK to replicate and specialize similar code instead of creating a more generalized library.
9. **Handle normal and worst cases separately as a rule.** Code should clearly special cases rather than use a more general algorithm that would remove the special cases.
10. **Split resources in a fixed way if in doubt.** Servers should be single purposed. For example, keep the database index and search index on separate machines. They can then be scaled independently and won't stomp on each other.
11. **Use static analysis if you can.** On check-in run stack analysis tools on code to find bugs and performance issues.
12. **Dynamic translation from a convenient representation to one that can be quickly interpreted.** For example, a Python domain specific language for tweet filtering was easy to program and could be directly translated to python bytecodes.
13. **Cache answers to expensive computations.** Self explanatory, but be careful of cache invalidation issues.
14. **When in doubt, use brute force.** It's better to complete a feature faster using a simple algorithm than it is to delay implementing a clever algorithm.
15. **Compute in background when possible.** Do as a little work as possible in the web server, queue it to background processes.
16. **Use Batch Processing if possible.** Loading individual data items is slow, load them in large batches.
17. **Shed load to control demand.** It's OK to have limits. Pick limits that make your software work without having to go through heroic efforts or change stacks.

Related Articles

- Hints for Computer System Design by Butler W. Lampson