Surviving the Big Rewrite:

Moving YELLOWPAGES.COM to Rails

John Straw
YELLOWPAGES.COM

What is YELLOWPAGES.COM?

- Part of AT&T
- A local search website, serving
  - 23 million unique visitors / month
  - 2 million searches / day
  - More than 48 million requests / day
  - More than 1500 requests / sec
  - 30 Mbit/sec (200 Mbit/sec from Akamai)
- Entirely Ruby on Rails for almost a year, following a Big Rewrite

When was the big rewrite?

Starting in late 2006 ...

- Several projects combined to become the big rewrite
  - Replacement of Java application server
  - Redesign of site look-and-feel
  - Many other wish-list projects, some of which were difficult to accomplish with existing application
- Project conception to completion: one year
- Development took just three months
- Project phases
  - **7/2006 - 12/2006:** Thinking, early preparation
  - **12/2006:** Rough architecture determination, kick-off
  - **1/2007 - 3/1/2007:** Technology research and prototypes, business rules exploration, UI design concepts
  - **3/1/2007 - 6/28/2007:** Site development and launch

Why a big rewrite?

Site design

- Site essentially unchanged since 2003 design
- Poor user experience demonstrated by usability testing
  - Lots of new browser windows confused users
  - Confusing controls and poor information layout

Application server change

- No useful platform support or improvements
- Session-heavy design hard to scale horizontally
- Unusable session migration features
- Platform design & misfeatures made SEO difficult

Code base

- Lots of code written by consultants 2004-2005
- Fundamental design problems
- Code extended largely by copy-and-modify since 11/2005 (to around 125K lines)
- No test code
- New features hard to implement
- Lots of code would be obsolete after site redesign and server replacement
- What remained was impossible to leverage

Rules / Requirements for new website

- Absolute control of urls
  - Maximize SEO crawlability
- No sessions: HTTP is stateless
  - Anything other than cookies is just self-delusion
  - Staying stateless makes scaling easier
- Be agile: write less code
  - Development must be faster
- Develop easy-to-leverage core business services
  - Eliminate current duplicated code
  - Must be able to build other sites or applications with common search, personalization and business review logic
  - Service-oriented architecture, with web tier utilizing common service tier

Who pulled off the big rewrite?

Team composition

- Cross-functional team of about 20 people
  - Ad products
  - Community features
  - Content
  - Development
  - Project management
  - Search
  - SEO
  - User experience (UX)
- Whole team sat together for entire project
- Lunch provided several days per week to foster team togetherness
- Team celebrations held for milestones

Benefits of team composition

- Diverse team helped ensure requirements weren't missed
  - Each team member had different perspectives about what was important
  - Each team member had to accept that only a small portion of his/her ax would be ground
- Core development team deliberately small
  - Never more than five developers -- four skilled developers can accomplish a lot
  - Cost of communication low on a small team -- especially important when working with new technology
  - Low management overhead

How did the team approach the big rewrite?

Exploration phase - development team

- Built initial Rails prototype -- small version of site
- Studied search code and search query logs
- Built prototype search service in Python
- Started new Rails prototype using search service and UX team proposed page designs
- Built a Django prototype using search service and UX team proposed page designs
- Evaluated and rejected EJB3/JBoss as a service tier platform
- Chose Rails for web tier and service tier

Why Rails?

- All considered Java frameworks didn't provide enough control of url structure
- Web tier platform choice came down to Rails vs. Django
- Rails best web tier choice due to
  - Better automated testing integration
  - More platform maturity
  - Clearer path to C if necessary for performance
  - Developer comfort and experience
- Originally thought that service tier would have to be Java/EJB3, but team decided to go with Rails top-to-bottom
  - Evaluation of EJB3 didn't show real advantages over Ruby or Python for our application
  - Reasons for choosing Rails for web tier applied equally to service tier
  - Advantage of having uniform implementation environment

Exploration phase - full team

- Requirements
  - Wish-list sharing
  - Lots of discussion about existing site and possible feature/behavior changes
  - UX team members given the task of building a catalog of existing site pages and features
- Communication with executive team
  - Extensive meetings summarizing current progress
  - Meetings caused distraction and were characterized by extensive low-level discussion
- Project in danger of stalling
  - Decision paralysis -- too many opinions
  - Over-ambitious expectations
  - Extremely short timeline

Development phase - Leadership

- Project lead appointed to take on the burden of decision-making and communication with executive team
- Agreement to freeze development on the existing site
- Establishment of decision-making rule: if it's not simple to decide how to change a current site behavior, don't change it
- Release schedule set
  - 04/26/2007 - "Friends and Family" beta
  - 05/17/2007 - Open beta
  - 06/28/2007 - Site launch

Development phase - Process

- Pages segmented into rough batches based on estimated importance and difficulty of implementation
- Multi-week page batch development cycle established
  - First week: batch wireframe development and sign-off
  - Second week: batch UI design and sign-off
  - Third week+: batch development
- Each week of cycle handled by different sub-team
- Multiple batch cycles run out-of-phase; each sub-team had a full pipeline

Development phase - Coding

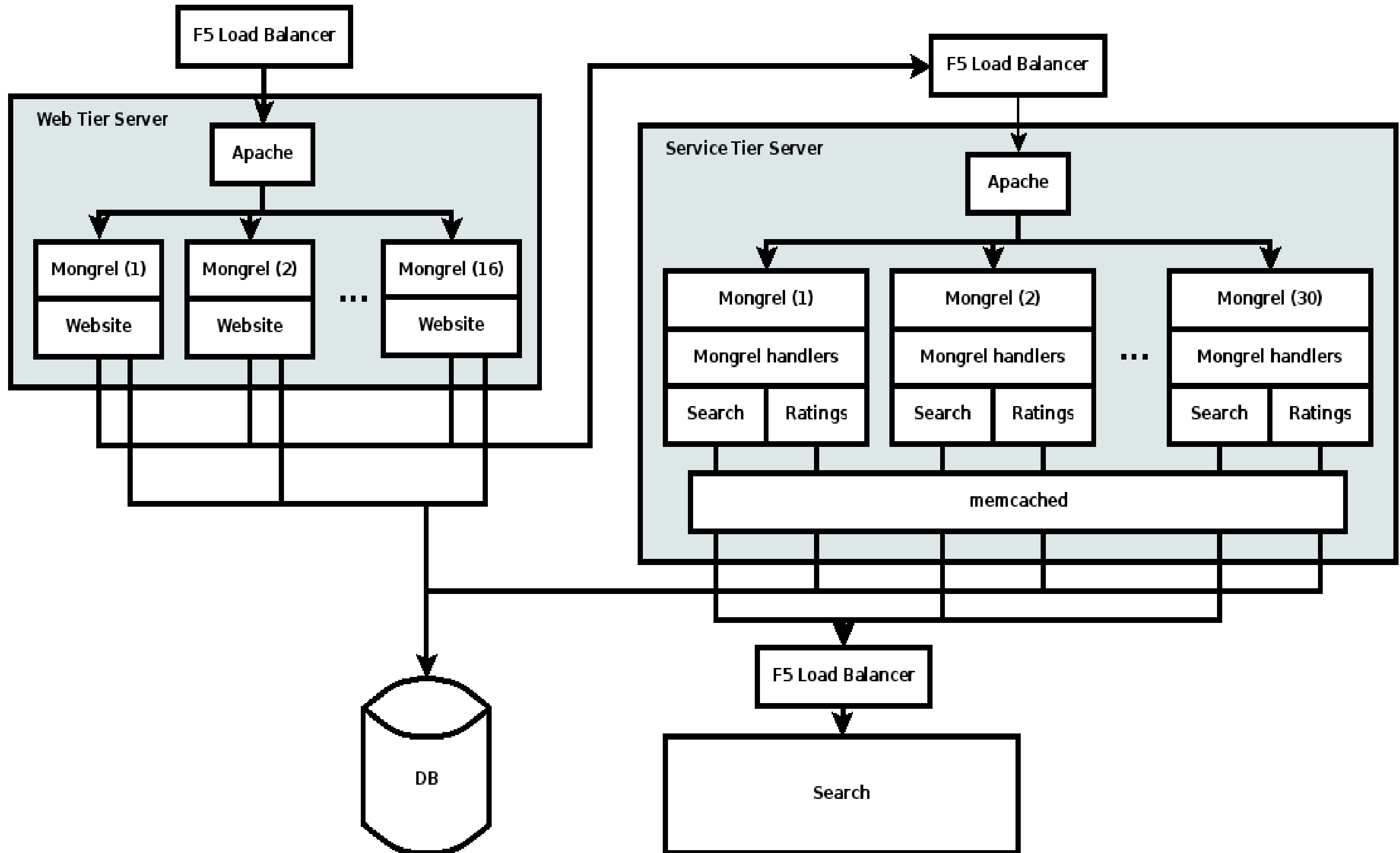
- Developed features needed for page batches and wired up HTML
- Deployed code often to integration/beta site -- visible progress
- Weekly milestones, with to-do lists managed in Basecamp
- Core development team stayed focused by outsourcing to other developers anything with manageable dependencies or requiring specific skills to do quickly
  - Static HTML/CSS coding of pages
  - Rewrite rules for legacy url translation
  - Performance evaluation for production deployment configuration

Development phase - Communication

- Early communication with sales team
  - Previous site redesign had nearly failed because of lack of communication with sales channels
  - Team member demonstrated beta site to groups of sales people
  - Talked to 20-40 salespeople a week, previewing proposed changes
  - Expected tour participants to discuss with their groups
- Project lead kept executive team happy
- Development lead kept CTO happy

What was built in the big rewrite?

Application design



Design features

- Stateless HTTP communication at all levels
- REST services returning JSON in service tier
- Memcached used extensively in service tier
- Production configuration
  - Acquired 25 machines of identical configuration for each data center (replacing 21 machines for existing site)
  - Performance testing to size out each tier, and determine how many mongrels
  - Used 2 machines in each data center for database servers

Performance

- Performance optimizations
  - Considered F5 vs. HAProxy vs. Swiftiply vs. localhost
  - Utilized mongrel_handlers in service tier application
  - Developed C library for parsing search cluster responses
  - Switched to Erubis in web tier
- Performance goals
  - Sub-second average home page load time
  - Sub 4-second average search page load time
  - Handle traffic without dying

Performance issues

- Database performance issues
  - Oracle doesn't like lots of connections
  - Machines inadequate to handle search load for ratings lookup
  - Additional caching added
- Web server performance issues
  - Slow page performance caused more by asset download times than speed of web framework
  - Worked through the Yahoo! performance guidelines
  - Minified and combined CSS and Javascript with asset_packager
  - Moved image serving to Akamai edge cache
  - Apache slow serving analytics tags -- moved to nginx for web tier
- Performance at launch was generally acceptable
- After web server & hosting changes performance better than previous site
- Extensive use of caching and elimination of obsolete queries lowered load on search cluster compared to previous site

Conclusion

- Project was exhausting, but a tremendous success
  - Site launched on schedule
  - Performance and stability were acceptable at launch
  - Team eventually wrote fewer than 20K lines of code and tests
- Keys to success
  - Small, talented development team
  - Careful evaluation of technology to choose platform compatible with application
  - Close communication among team members with diverse viewpoints allowed us capture requirements without formalism
  - Freeze on existing site prevented us from having to hit moving target
  - "Don't change things which aren't simple to decide" rule prevented decision paralysis
  - Frequent beta site updates allowed early, visible communication of progress and direction
  - CTO not only allowed us to bet on Rails, but was excited about the idea