# High Scalability - High Scalability - eBay Architecture

highscalability.com

Tuesday, May 27, 2008 at 8:13AM

**Update 2:** EBay's Randy Shoup spills the secrets of how to service hundreds of millions of users and over two billion page views a day in Scalability Best Practices: Lessons from eBay on InfoQ. The practices: Partition by Function, Split Horizontally, Avoid Distributed Transactions, Decouple Functions Asynchronously, Move Processing To Asynchronous Flows, Virtualize At All Levels, Cache Appropriately.
**Update:** eBay Serves 5 Billion API Calls Each Month. Aren't we seeing more and more traffic driven by mashups composed on top of open APIs? APIs are no longer a bolt on, they are your application. Architecturally that argues for implementing your own application around the same APIs developers and users employ. Who hasn't wondered how eBay does their business? As one of the largest most loaded websites in the world, it can't be easy. And the subtitle of the presentation hints at how creating such a monster system requires true engineering: Striking a balance between site stability, feature velocity, performance, and cost.

You may not be able to emulate how eBay scales their system, but the issues and possible solutions are worth learning from.

Site: http://ebay.com

## Information Sources

- The eBay Architecture - Striking a balance between site stability, feature velocity, performance, and cost.
- Podcast: eBay's Transactions on a Massive Scale
- Dan Pritchett on Architecture at eBay interview by InfoQ

## Platform

- Java
- Oracle
- WebSphere, servlets
- Horizontal Scaling
- Sharding
- Mix of Windows and Unix

## What's Inside?

This information was adapted from Johannes Ernst's Blog

### The Stats

- On an average day, it runs through 26 billion SQL queries and keeps tabs on 100 million items available for purchase.
- 212 million registered users, 1 billion photos
- 1 billion page views a day, 105 million listings, 2 petabytes of data, 3 billion API calls a month
- Something like a factor of 35 in page views, e-mails sent, bandwidth from June 1999 to Q3/2006.
- 99.94% availability, measured as "all parts of site functional to everybody" vs. at least one part of a site not functional to some users somewhere
- The database is virtualized and spans 600 production instances residing in more

than 100 server clusters.

- 15,000 application servers, all J2EE. About 100 groups of functionality aka “apps”. Notion of a “pool”: “all the machines that deal with selling”…

## The Architecture

- Everything is planned with the question “what if load increases by 10x”. Scaling only horizontal, not vertical: many parallel boxes.
- Architectures is strictly divided into layers: data tier, application tier, search, operations,
- Leverages MSXML framework for presentation layer (even in Java)
- Oracle databases, WebSphere Java (still 1.3.1)
- Split databases by primary access path, modulo on a key.
- Every database has at least 3 on-line databases. Distributed over 8 data centers
- Some database copies run 15 min behind, 4 hours behind
- Databases are segmented by function: user, item account, feedback, transaction, over 70 in all.
- No stored procedures are used. There are some very simple triggers.
- Move cpu-intensive work moved out of the database layer to applications applications layer: referential integrity, joins, sorting done in the application layer! Reasoning: app servers are cheap, databases are the bottleneck.
- No client-side transactions. no distributed transactions
- J2EE: use servlets, JDBC, connection pools (with rewrite). Not much else.
- No state information in application tier. Transient state maintained in cookie or scratch database.
- App servers do not talk to each other — strict layering of architecture
- Search, in 2002: 9 hours to update the index running on largest Sun box available — not keeping up.
- Average item on site changes its search data 5 times before it is sold (e.g. price), so real-time search results are extremely important.

- "Voyager": real-time feeder infrastructure built by eBay.. Uses reliable multicast from primary database to search nodes, in-memory search index, horizontal segmentation, N slices, load-balances over M instances, cache queries.

## Lessons Learned

- **Scale Out, Not Up**– Horizontal scaling at every tier.– Functional decomposition.

- **Prefer Asynchronous Integration**– Minimize availability coupling.– Improve scaling options.

- **Virtualize Components**– Reduce physical dependencies.– Improve deployment flexibility.

- **Design for Failure**– Automated failure detection and notification.– "Limp mode" operation of business features.

- **Move work out of the database into the applications because the database is the bottleneck**. Ebay does this in the extreme. We see it in other architecture using caching and the file system, but eBay even does a lot of traditional database operations in applications (like joins).

- **Use what you like and toss what you don't need**. Ebay didn't feel compelled to use full blown J2EE stack. They liked Java and Servlets so that's all they used. You don't have to buy into any framework completely. Just use what works for you.

- **Don't be afraid to build solutions that meet and evolve with your needs**. Every off the shelf solution will fail you at some point. You have to go the rest of the way on your own.

- **Operational controls become a larger and larger part of scalability as you grow**. How do you upgrade, configure, and monitor thousands of machines will running a live system?

- **Architectures evolve.** You need to be able to change, refine, and develop your new system while keeping your existing site running. That’s the primary challenge of any growing website.

- **It’s a mistake to worry too much about scalability from the start**. Don’t suffer from paralysis by analysis and worrying about traffic that may never come.

- **It’s also a mistake not to worry about scalability at all**. You need to develop an organization capable of dealing with architecture evolution. Understand you are never done. Your system will always evolve and change. Build those expectations and capabilities into your business from the start. Don’t let people and organizations be why your site fails. Many people will think the system should be perfect from the start. It doesn’t work that way. A good system is developed overtime in response to real issues and concerns. Expect change and adapt to change.

highscalability.com ➔