The eBay Architecture

Striking a balance between site stability, feature velocity, performance, and cost

SD Forum 2006

Presented By: Randy Shoup and Dan Pritchett
Date: November 29, 2006
What we’re up against

- eBay manages …
  - Over 212,000,000 registered users
  - Over 1 Billion photos
  - eBay users worldwide trade more than $1590 worth of goods every second
  - eBay averages over 1 billion page views per day
  - At any given time, there are approximately 105 million listings on the site
  - eBay stores over 2 Petabytes of data – over 200 times the size of the Library of Congress!
  - The eBay platform handles 3 billion API calls per month

- In a dynamic environment
  - 300+ features per quarter
  - We roll 100,000+ lines of code every two weeks

- In 33 countries, in seven languages, 24x7

>26 Billion SQL executions/day!

Over ½ Million pounds of Kimchi are sold every year!
eBay’s Exponential Growth

105 Million Listings

212 Million Users
• Our site is our product. We change it incrementally through implementing new features.
• Very predictable development process – trains leave on-time at regular intervals (weekly).
• Parallel development process with significant output -- 100,000 LOC per release.
• Always on – over 99.94% available.

All while supporting a 24x7 environment
Systemic Requirements

- **Availability**
  - Reliability
  - Massive Scalability
  - Security
  
  Enable seamless growth

- **Maintainability**
  - Faster Product Delivery
  
  Deliver quality functionality at accelerating rates

- **Architect for the future**
  - 10X Growth
  
  Enable rapid business innovation
Architectural Lessons

• 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.
Ongoing Platform Evolution...

Registered Users

eBay architecture versions

<table>
<thead>
<tr>
<th>Year</th>
<th>Version</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>V1</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>Q3</td>
<td>212M</td>
</tr>
</tbody>
</table>
- Built over a weekend in Pierre Omidyar’s living room in 1995
- System hardware was made up of parts that could be bought at Fry's
- Every item was a separate file, generated by a Perl script
- No search functionality, only category browsing

<table>
<thead>
<tr>
<th>Domain</th>
<th>Version</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculus.ebay.com</td>
<td></td>
<td>FreeBSD, Apache, Perl, GDBM</td>
</tr>
<tr>
<td>trig.ebay.com</td>
<td></td>
<td>FreeBSD, Apache, Perl, GDBM</td>
</tr>
<tr>
<td>thompson.ebay.com</td>
<td></td>
<td>FreeBSD, Apache, Perl, GDBM</td>
</tr>
<tr>
<td>thomson.ebay.com</td>
<td></td>
<td>FreeBSD, Perl, GDBM</td>
</tr>
</tbody>
</table>

This system maxed out at 50,000 active items
3-tiered conceptual architecture (separation of bus/pres and db access tiers)
2-tiered physical implementation (no application server)
C++ Library (eBayISAPI.dll) running on IIS on Windows
Microsoft index server used for search
Items migrated from GDBM to an Oracle database on Solaris
- Servers grouped into pools (small soldiers)
- Resonate used for front end load balancing and failover
- Search functionality moved to the Thunderstone indexing system
- Back-end Oracle database server scaled vertically to a larger machine (Sun E10000)
• Second Database added for failover
• CGI pools, Listings, Pages, and Search continued to scale horizontally

However ...

By November 1999, the database servers approached their limits of physical growth.
• Database "split" technology.
• Logically partition database into separate instances.
• Horizontal scalability through 2000, but not beyond.
• Horizontal scalability through database splits
• Items split by category
• SPOF elimination

December, 2002
Now that we have the Database taken care of…

- Application Server
  - Monolithic 2-tier Architecture
  - 3.3 Million Line C++ ISAPI DLL (150MB binary)
  - Hundreds of developers, all working on the same code
  - Hitting compiler limits on number of methods per class (!!)
V3 – Replace C++/ISAPI with Java \textit{2002-present}

- Re-wrote the entire application in J2EE application server framework
  - Gave us a chance to architect the code for reuse and separation of duties
- Leveraged the MSXML framework for the presentation layer
  - Minimizing the development cost for migration
- Implemented a development kernel as a foundation for programmers
  - Allowed for rapid training and deployment of new engineers
Scaling the Data Tier
Scaling the Data Tier: Overview

• Spread the Load
  – Segmentation by function.
  – Horizontal splits within functions.

• Minimize the Work
  – Limit in database work

• The Tricks to Scaling
  – How to survive without transactions.
  – Creating alternate database structures.
Scaling the Data Tier: Functional Segmentation

- Segment databases into functional areas
  - User hosts
  - Item hosts
  - Account hosts
  - Feedback hosts
  - Transaction hosts
  - And about 70 more functional categories

- Rationale
  - Partitions data by different scaling / usage characteristics
  - Supports functional decoupling and isolation
Scaling the Data Tier: Horizontal Split

- Split databases horizontally by primary access path.

- Different patterns for different use cases
  - Write Master/Read Slaves
  - Segmentation by data; Two approaches
    - Modulo on a key, typically the primary key.
      - Simple data location if you know the key
      - Not so simple if you don’t.
    - Map to data location
      - Supports multiple keys.
      - Doubles reads required to locate data.
      - SPOF elimination on map structure is complex.

- Rationale
  - Horizontal scaling of transactional load.
  - Segment business impact on database outage.
Scaling the Data Tier: Logical Database Hosts

- Separate Application notion of a database from physical implementation
- Databases may be combined and separated with no code changes
- Reduce cost of creating multiple environments (Dev, QA, …)
Scaling the Data Tier: Minimize DB Resources

- No business logic in database
  - No stored procedures
  - Only very simple triggers (default value population)

- Move CPU-intensive work to applications
  - Referential Integrity
  - Joins
  - Sorting

- Extensive use of prepared statements and bind variables
Scaling the Data Tier: Minimize DB Transactions

- Auto-commit for vast majority of DB writes
- Absolutely no client side transactions
  - Single database transactions managed through anonymous PL/SQL blocks.
  - No distributed transactions.
- How do we pull it off?
  - Careful ordering of DB operations
  - Recovery through
    - Asynchronous recovery events
    - Reconciliation batch
    - Failover to async flow
- Rationale
  - Avoid deadlocks
  - Avoid coupling availability
  - Update concurrency
  - Seamless handling of splits
Scaling the Application Tier
Scaling the Application Tier – Overview

• Spread the Load
  – Segmentation by function.
  – Horizontal load-balancing within functions.

• Minimize dependencies
  – Between applications
  – Between functional areas
  – From applications to data tier resources

• Virtualize data access
Scaling the Application Tier – Massively Scaling J2EE

- **Step 1 - Throw out most of J2EE**
  - eBay scales on servlets and a rewritten connection pool.

- **Step 2 – Keep Application Tier Completely Stateless**
  - No session state in application tier
  - Transient state maintained in cookie or scratch database

- **Step 3 – Cache Where Possible**
  - Cache common metadata across requests, with sophisticated cache refresh procedures
  - Cache reload from local storage
  - Cache request data in ThreadLocal
Scaling the Application Tier – Tiered Application Model

- Strictly partition application into tiers
  - Presentation
  - Business
  - Integration

Presentation Tier

- XSL
- Command (View)
- AO/AOF (View)

Business Tier

- BO/BOF

Integration Tier

- DO/DAO

XML Model Building Logic
Business Logic
Data Access Layer (DAL)
Scaling the Application Tier – Data Access Layer (DAL)

• What is the DAL?
  – eBay’s internally-developed pure Java OR mapping solution.
  – All CRUD (Create Read Update Delete) operations are performed through DAL’s abstraction of the data.
  – Enables horizontal scaling of the Data tier without application code changes

• Dynamic Data Routing abstracts application developers from
  – Database splits
  – Logical / Physical Hosts
  – Markdown
  – Graceful degradation

• Extensive JDBC Prepared Statements cached by DataSources
Scaling the Application Tier – Vertical Code Partitioning

• Partition code into functional areas
  • Application is specific to a single area (Selling, Buying, etc.)
  • Domain contains common business logic across Applications

• Restrict inter-dependencies
  • Applications depend on Domains, not on other Applications
  • No dependencies among shared Domains
Scaling the Application Tier – Functional Segmentation

- Segment functions into separate application pools
  - Minimizes/isolates DB dependencies
  - Allows for parallel development, deployment, and monitoring

ViewItem Pool
http://cgi.ebay.com ...

Load Balancing

IIS

CGI0

Load Balancing

IIS

IIS

Load Balancing

AS

AS

AS

Load Balancing

User

Acct

SYI Pool
http://cgi5.ebay.com ...

Load Balancing

IIS

IIS

Load Balancing

AS

AS

AS

Load Balancing

Caty1

Caty20+
Scaling the Application Tier – Platform Decoupling

• Domain Partitioning for Deployment
  – Decouple non-transactional domains from transactional flows
    • Search and billing domains are not required in transaction processing.
    • Fraud domain is required but easier to manage as separate deployment.
  – Integrate with a combination of asynchronous EDA and synchronous SOA patterns.
Scaling Search
Scaling Search – Overview

• In 2002, eBay search had reached its limits
  – Cost of scaling third-party search engine had become prohibitive
  – 9 hours to update the index
  – Running on largest systems vendor sold – and still not keeping up

• eBay has unique search requirements
  – Real-time updates
    • Update item on any change (list, bid, sale, etc.)
    • Users expect changes to be visible immediately
  – Exhaustive recall
    • Sellers notice if search results miss any item
    • Search results require data (“histograms”) from every matching item
  – Flexible data storage
    • Keywords
    • Structured categories and attributes

• No off-the-shelf product met these needs
Scaling Search – Voyager

- Real-time feeder infrastructure
  - Reliable multicast from primary database to search nodes

- Real-time indexing
  - Search nodes update index in real time from messages

- In-memory search index

- Horizontal segmentation
  - Search index divided into N slices ("columns")
  - Each slice is replicated to M instances ("rows")
  - Aggregator parallelizes query over all N slices, load-balances over M instances

- Caching
  - Cache results for highly expensive and frequently used queries
Scaling Operations
Scaling Operations – Code Deployment

• Demanding Requirements
  – Entire site rolled every 2 weeks
  – All deployments require staged rollout with immediate rollback if necessary.
  – More than 100 WAR configurations.
  – Dependencies exist between pools during some deployment operations.
  – More than 15,000 instances across eight physical data centers.

• Rollout Plan
  – Custom application that works from dependencies provided by projects.
  – Creates transitive closure of dependencies.
  – Generates rollout plan for Turbo Roller.

• Automated Rollout Tool (“Turbo Roller”)
  – Manages full deployment cycle onto all application servers.
  – Executes rollout plan.
  – Built in checkpoints during rollout, including approvals.
  – Optimized rollback, including full rollback of dependent pools.
Scaling Operations – Monitoring

- Centralized Activity Logging (CAL)
  - Transaction oriented logging per application server
    - Transaction boundary starts at request. Nested transactions supported.
    - Detailed logging of all application activity, especially database and other external resources.
    - Application generated information and exceptions can be reported.
  - Logging streams gathered and broadcast on a message bus.
    - Subscriber to log to files (1.5TB/day)
    - Subscriber to capture exceptions and generate operational alerts.
    - Subscriber for real time application state monitoring.
  - Extensive Reporting
    - Reports on transactions (page and database) per pool.
    - Relationships between URL’s and external resources.
    - Inverted relationships between databases and pools/URL’s.
    - Data cube reporting on several key metrics available in near real time.
Recap

Enabling seamless growth
- Massive Database and Code Scalability

Delivering quality functionality at accelerating rates
- Further streamline and optimize the eBay development model

Enabling rapid business innovation

Availability
Reliability
Massive Scalability
Security

Maintainability
Faster Product Delivery

Architecting for the future
10X Growth