

N-Tier High Scale Database Application

Jamie Hogleund

<http://www.genigate.com/>

Formerly, XML Global Technologies Inc.

ABSTRACT

This is a case story of a confidential database application used by law enforcement. Although the sensitive nature of the project forbids disclosure of specific details, the principles of the design architecture may be applied to a wide range of enterprise level problems.

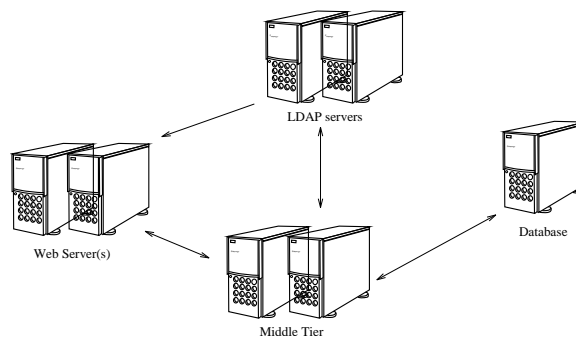
1. Product Ownership

Although the system was designed *for* law enforcement, it was owned and operated by private industry. Law enforcement rented the package in much the same way one might rent web hosting or remote e-commerce solutions.

I was responsible for the initial design and development of the package.

The system was designed for the ultimate deployment on a nation-wide basis. Downtime would be extremely expensive, therefore measures were taken to minimize this possibility.

2. Redundant Design Model



The system consisted of a number of machines configured in the classic *n-tier* model. With status notifications and system configuration handled by LDAP. The database could be either *postgresql* or IBM's *DB2*.

2.1. Everything as a Service

When designing high scale redundant systems, it is helpful to picture each component in terms of multiples.

The analogy of a *service* rather than a *server* is useful in this context. For example, email is a service consisting of multiple SMTP servers. Individual SMTP servers may go offline without affecting email *service* as a whole.

2.2. Notification of System Changes

One problem with multiple sub-systems is that of maintenance and upgrades. We needed a means of propagating status changes (such as new hardware being brought online) throughout the system.

With built in redundancy, LDAP was a natural fit. LDAP had already solved some problems related to user management and could easily perform this task as well. †

Other software could then access the *LDAP service* for configuration, resulting in near instant awareness of changes.

This worked out well, configuration changes could be performed in "real time" **without** addressing servers individually

2.3. Middle Tier

The business logic manifested itself in the form of a *middle tier*. The middle tier was responsible for any activity involving database access, report generation and basic user management.

By partitioning the workload in this manner, we were able to keep the web servers light, conserve database resources and achieve a clear division in functionality.

2.3.1. Report Generation

The middle tier was responsible for composing several XML formatted reports based on an aggregate of information pulled from various confidential sources.

2.3.2. Database Access

While the middle tier approach supported a redundant database model, resource limitations prevented its implementation. The database would be the *only* central point of failure.

All database access was performed by the middle tier. The human involvement with this system involved primarily table *inserts*, therefore indexes were *reduced* to minimize the database overhead that would normally be applied toward index maintenance.

The system started out using *postgresql* and was later moved to IBM's *DB2* product.

To facilitate SQL porting and tuning, all SQL statements were kept in an external file and loaded into the system. This novel approach allowed us to *tune* the SQL as the tables grew in size.

As a side effect of this approach, the system could easily be ported to different databases by merely adjusting the external SQL statements and tweaking a few routines related to time handling. For example, moving from *postgresql* to *DB2* took only two hours of development time.

2.4. Web Servers

The front end of the system consisted of mod_perl applications running on apache web servers.

This layer was deliberately limited to user interface logic, all database access was performed via the middle tier *service*.

This approach preserved the continuity of the system and maintained the logical division of tasks.

2.5. Challenges

The largest challenge was in the area of distributed storage.

Simple operations having to do with I/O (such as opening a file) became incredibly complex, such information needed to be replicated across a dynamic number of machines.

Mysql might also have been used for this purpose.

3. Project Status

Toward the end of the company's life span, a corporate shift from Perl to Java had taken place. While I did work on a port of the front end to Java Servlet technology, a decision was made to port the middle tier to EJB. This rendered the servlets I had been working on incompatible.

The original system continued to function after I left the company, allowing the new work force (primarily Java programmers) to assume command of the project.

At last communication, the system was sold to another company and ported to Java.

4. Conclusion

The system enabled law enforcement to conduct operations at greatly reduced cost with unparalleled speed and efficiency.

Some of the staff members claimed tasks once taking a period of months now took one day and in extreme cases, took only *minutes*.

Additionally, this system enabled them to perform operations that were previously impossible.

The only regret I have is that Java would have been a more suitable technology for this application.