

Performance Considerations for Web Applications

By Dr. Paul Dorsey & Michael Rosenblum, Dulcian, Inc.

Many of the performance tuning techniques applied to client/server applications that consisted of rewriting poorly written SQL code and tuning the database itself are not helpful when dealing with Web applications that are frequently unaffected by these performance improvement approaches. This tip describes some of the most common problem areas when dealing with Web application performance issues.

Typical Web Application Process Flow

Improving the performance of a slow Web application requires examination of the entire system, not just the database. A typical 3-tier Web application structure is shown in Figure 1.

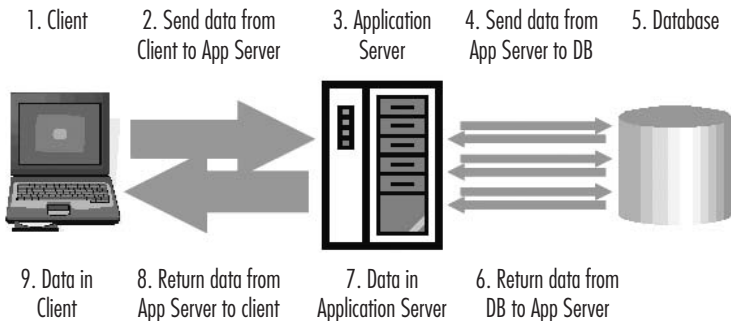


Figure 1: Web Application Process Flow



As shown in Figure 1, there are numerous possible places for Web applications to experience bottlenecks or performance killers as described in the following nine-step process:

- Step 1: Code and operations executed on the client machine
- Step 2: Transmission of client request to the application server
- Step 3: Code in the application server executed as a formulation of the client request to retrieve information from the database
- Step 4: Transmission of request from application server to the database
- Step 5: Database reception, processing and preparation of return information to the application server
- Step 6: Transmission over internal network of information from database to the application server
- Step 7: Application server processing of database response and preparation of response transmission to the client machine
- Step 8: Transmission of data from the application server to the client machine
- Step 9: Client machine processing of returned request and rendering of application page in browser

Web Application Performance Problem Areas

Traditional tuning techniques only help with Step 5 and ignore all of the other eight places where performance can degrade. This section describes how problems can occur at each step of the process.



Step 1. Client Machine Performance Problems

The formulation of a request in the client is usually the least likely source of system performance problems. However, it should not be dismissed entirely. Using many modern AJAX architectures, it is possible to place so much code in the client that a significant amount of time is required before the request is transmitted to the application server.

This is particularly true for underpowered client machines with inadequate memory and slow processors.

Step 2. Client-to-Application Server Transmission Problems

Like the client machine itself, the transmission time between the client machine and the application server is a less common cause of slowly performing Web applications. However, if attempting to transmit a large amount of information, the time required to do so over the Internet may be affected. For example, uploading large files or transmitting a large block of data may slow down performance.

Step 3. Application Server Performance Problems

The application server itself rarely causes significant performance degradation. For computationally intensive applications such as large matrix inversions for linear programming problems, some performance slowdowns can occur, but this is less likely to be a significant factor in poorly performing applications.

Step 4. Application Server to Database Transmission Problems

Transmission of data from the application server to the database with 1 GB or better transmission speeds might lead you to ignore this step in the process. It is not the time needed to move data from the application server to the database that causes performance degradation, but the high number of transmission requests. The trend in current Web development is to make applications database-independent. This sometimes results in a single request from a client requiring many requests from the application server to the




database in order to fulfill. What needs to be examined and measured is the number of roundtrips made from the application server to the database.

Inexpert developers may create routines that execute so many roundtrips that there is little tuning that a DBA can do to yield reasonable performance results. It is not unusual for a single request from the client to generate hundreds (if not thousands) of round trips from the application server to the database before the transmission is complete. Java developers who think of the database as nothing more than a place to store persistent copies of their classes use Getters and Setters to retrieve and/or update individual attributes of objects. This type of development can generate a round trip for every attribute of every object in the database, which means that inserting a row into a table with 100 columns results in a single Insert followed by 99 Update statements. Retrieving this record from the database then requires 100 independent queries.

In the application server, identifying performance problems involves counting the number of transmissions made. The accumulation of time spent making round trips is one of the most common places where Web application performance can suffer.

Step 5. Database Performance Problems

In the database itself, it is important to look for the same things that cause client/server applications to run slowly. However, additional Web application features can cause other performance problems in the database. Most Web applications are stateless, meaning that each client request is independent. Developers do not have the ability to use package variables that persist over time. Consequently, when a user logs into an application, he or she will be making multiple requests within the context of the sign-on operation. The information pertaining to that session must be retrieved at the beginning of every request and persistently stored at the end of every request. Depending upon how this persistence is handled in the database, a single table may generate massive I/O demands resulting in redo logs



full of information that may cause contention on tables where session information is stored.

Step 6. Database-to-Application Server Transmission Problems

Transferring information from the database back to the application server (similar to Step 4) is usually not problematic from a performance standpoint. However, performance can suffer when a Java program requests a single record from a table. If the entire table contents are brought into the middle tier and then filtered to find the appropriate record, performance will be slow. The application can perform well as long as data volumes are small. As data accumulates, the amount of information transferred to the application server becomes too large, thus affecting performance.

Step 7. Application Server Processing Performance Problems

Processing the data from the database can be resource-intensive. Many database-independent Java programmers minimize work done in the database and execute much of the application logic in the middle tier. In general, complex data manipulation can be handled much more efficiently with database code. Java programmers should minimize information returned to the application server and, where convenient, use the database to handle computations.

Step 8. Application Server-to-Client Machine Transmission Problems

This area is one of the most important for addressing performance problems and often receives the least attention. Industry standards often assume that everyone has access to high performance client machines so that the amount of data transmitted from the application server to the client is irrelevant. As the industry moves to Web 2.0 and AJAX, very rich UI applications create more and more bloated screens of 1 MB or more. Some of the AJAX partial page refresh capabilities mitigate this problem somewhat (100-200K). Since most Web pages only need to logically transmit an amount of information requiring 5K or less, the logical round trips on an open page should be measured in tens or hundreds of characters rather than megabytes. Transmission between the application server and the



client machine can be the most significant cause of poor Web application performance. If a Web page takes 30 seconds to load, even if it is prepared in five rather than 10 seconds, users will not experience much of a benefit. The amount of information being sent must be decreased.

Step 9. Client Performance Problems

How much work does the client need to do to render a Web application page? This area is usually not a performance killer, but it can contribute to poor performance. Very processing-intensive page rendering can result in poor application performance.

Conclusions

There is much more to tuning a Web application than simply identifying slow database queries. Changing database and operating system parameters will only go so far. The most common causes of slow performance are as follows.

Excessive round trips from the application server to the database - Ideally, each UI operation should require exactly one round trip to the database. Sometimes, the framework (such as ADF) will require additional round trips to retrieve and persist session data. Any UI operation requiring more than a few round trips should be carefully investigated.

Large pages sent to the client - Developers often assume that all of the system users have high-speed Internet connections. Everyone has encountered slow loading Web pages taking multiple seconds to load. Once in a while, these delays are not significant. However, this type of performance degradation (waiting three seconds for each page refresh) in an application such as a data entry intensive payroll application is unacceptable. Applications should be architected to take into account the slowest possible network to support when testing the system architecture for suitability in slower environments.



Performing operations in the application server that should be

done in the database - For large, complex systems with sufficient data volumes, complete database independence is very difficult to achieve. The more complex and data-intensive a routine, the more likely it is that it will perform much better in the database. For example, the authors encountered a middle tier Java routine that required 20 minutes to run. This same routine ran in 2/10 of a second when refactored in PL/SQL and moved to the database.

Poorly written SQL and PL/SQL routines - In some organizations, this may be the No. 1 cause of slowly performing Web applications. This situation often occurs when Java programmers are also expected to write a lot of SQL code. In most cases, the performance degradation is not caused by a single slow running routine but a tendency to fire off more queries than are needed.

Keeping all nine of the potential areas for encountering performance problems in mind and investigating each one carefully can help to identify the cause of a slowly performing Web application and point to ways in which that performance can be improved.



■ ■ ■ About the Author

Dr. Paul Dorsey is the founder and president of Dulcian, Inc. an Oracle consulting firm specializing in business rules and Web-based application development. He is the chief architect of Dulcian's Business Rules Information Manager (BRIM[®]) tool. Dr. Dorsey is the co-author of seven Oracle Press books on Designer, Database Design, Developer, and JDeveloper, which have been translated into nine languages as well as the Wiley Press book *PL/SQL for Dummies*. Dr. Dorsey is an Oracle ACE Director. He is president emeritus of NYOUG and the associate editor of the International Oracle User Group's *SELECT Journal*. In 2003, Dr. Dorsey was honored by ODTUG as volunteer of the year and as Best Speaker (Topic & Content) for the 2007 conference, in 2001 by IOUG as volunteer of the year and by Oracle as one of the six initial honorary Oracle 9i Certified Masters. He is also the founder and chairperson of the ODTUG Symposium, currently in its ninth year. Dr. Dorsey's submission of a Survey Generator built to collect data for The Preeclampsia Foundation was the winner of the 2007 Oracle Fusion Middleware Developer Challenge and Oracle selected him as the 2007 PL/SQL Developer of the Year.

Michael Rosenblum is a development DBA at Dulcian, Inc. He is responsible for system tuning and application architecture. He supports Dulcian developers by writing complex PL/SQL routines and researching new features. Rosenblum is the co-author of *PL/SQL for Dummies* (Wiley Press, 2006). He is a frequent presenter at various regional and national Oracle user group conferences. In his native Ukraine, he received the scholarship of the President of Ukraine, a master's degree in Information Systems, and a diploma with honors from the Kiev National University of Economics.