Craig S. Mullins

Return to Home Page

Vol. 14, No. 1 (Spring 2007)

From IDUG Solutions Journal...

The Buffer Pool

DB2 Database Performance Fundamentals

By Craig S. Mullins

Applications that access relational databases are only as good as the performance they achieve. And every user wants their software to run as fast as possible. As such, performance tuning and management is one of the biggest demands on the DBA's time. When asked what is the single most important or stressful aspect of their job, DBAs typically respond "assuring optimal performance." Indeed, a recent Forrester Research survey indicates that performance and troubleshooting tops the list of most challenging DBA tasks.

Handling performance problems should be an enterprise-wide endeavor. And most organizations monitor and tune the performance of their entire IT infrastructure encompassing servers, networks, applications, desktops, and databases. However, the task of enterprise performance management frequently becomes the job of the DBA group. Anyone who has worked as a DBA for any length of time knows that the DBMS is usually "guilty until proven innocent." Every performance problem gets blamed on the database regardless of its true source cause. DBAs need to be able research and decipher the true cause of all performance degradation, if only to prove that it is not caused by a database problem.

As such, DBAs must be able to understand at least the basics of the entire IT infrastructure, but also need to have many friends who are experts in other related fields (such as networking, operating systems, communication protocols, etc.). Possessing a sound understanding of the IT infrastructure enables DBAs to respond effectively when performance problems arise. Event-driven tools exist on the market that can make performance management easier by automatically invoking pre-defined actions when specific alerts are triggered. For example, an alert can be set to proactively reorganize a database when it reaches its storage capacity. And other tools exist that can ease the burden of performance management and analysis.

But many of the supposedly proactive steps taken against completed applications in production are truly mostly reactive. Let's face it, DBAs are often too busy taking care of the day-to-day tactical database administration tasks to proactively monitor and tune their systems to the degree they wish they could. You know the drill – a user calls with a response time problem... a table space runs out of space to expand... the batch window extends into the day… someone submitted that "query from hell" that just won't stop running. Those of you in the trenches can relate — you've been there; done that.

Defining Database Performance

All of this discussion is useful, but it begs the question: just what do we mean by the term database performance? We need a firm definition of database performance before we can learn ways to plan for efficiency. Think, for a moment, of database performance using the familiar concepts of supply and demand. Users demand information from the database. The DBMS supplies information to those requesting it. The rate at which the DBMS supplies the demand for information can be termed "database performance."

Five factors influence database performance: workload, throughput, resources, optimization, and contention.

The *workload* that is requested of the DBMS defines the demand. It is a combination of online transactions, batch

jobs, ad hoc queries, data warehousing analysis, and system commands directed through the system at any given time. Workload can fluctuate drastically from day to day, hour to hour, and even minute to minute. Sometimes workload can be predicted (such as heavy month-end processing of payroll, or very light access after 5:30 p.m., when most users have left for the day), but at other times it is unpredictable. The overall workload has a major impact on database performance.

Throughput defines the overall capability of the computer to process data. It is a composite of I/O speed, CPU speed, parallel capabilities of the machine, and the efficiency of the operating system and system software. The hardware and software tools at the disposal of the system are known as the *resources* of the system. Examples: database kernel, disk space, cache controllers, and microcode.

The fourth defining element of database performance is *optimization*. All types of systems can be optimized, but relational databases are unique in that query optimization is primarily accomplished internal to the DBMS. However, there are many other factors that need to be optimized (SQL formulation, database parameters, etc.) to enable the database optimizer to create the most efficient access paths.

When the demand (workload) for a particular resource is high, contention can result. *Contention* is the condition in

which two or more components of the workload are attempting to use a single resource in a conflicting way (for example, dual updates to the same piece of data). As contention increases, throughput decreases.

Therefore, database performance can be defined as the optimization of resource use to increase throughput and minimize contention, enabling the largest possible workload to be processed. Of course, I do not advocate managing database performance in a vacuum. In addition, applications regularly communicate with other subsystems and components of the IT infrastructure. Each of these must also be factored into the overall performance planning of your organization. But it is wise to place limits on the actual responsibility for tuning outside the scope of this definition. If it is not defined above, it probably requires expertise outside the scope of database administration. Therefore, performance management tasks not covered by the above description should be handled by someone other than the DBA — or at a minimum shared between the DBA and other technicians.

A Basic Database Performance Roadmap

Now that we have defined what we mean by database performance, we need to forge a basic plan to ensure that database performance management and analysis is accomplished at our site.

Every database application, at its core, requires three components to operate:

- the system,
- the database, and
- the application.

This is true for any application that accesses data in a database. To improve performance, the administrator must be able to monitor and tune each of these components. But what, exactly, are each of these components?

The *system* consists of the system software and hardware required for the application to provide service. This includes the computer itself, its disk subsystems, network connections, and all peripherals. From a software perspective the system includes the operating system, the file system, the DBMS, networking protocols, and any related middleware, such as transaction processors or message queues.

To deliver system performance, the DBA must have the resources to monitor, manage, and optimize the performance of these disparate pieces of hardware and software. Some of the tasks required for system tuning include properly allocating and managing memory structures (such as buffer pools and program cache area), managing storage, integrating the DBMS with other system software, properly

using database logs, and coordinating the DBMS's operating system resources. Additionally, DBAs must control the installation, configuration, and migration of the DBMS software. If the system isn't performing properly, everything that uses the system will perform poorly. In other words, a poorly performing system affects every database application.

Memory structures, specifically data buffers, are probably the most important aspect of system performance with respect to database systems. When data can be accessed from memory instead of from disk overall performance can be improved by several orders of magnitude.

Indeed, every DBMS works best when it uses memory efficiently: this is where autonomic tuning is proving to be beneficial. By allowing the system to expand, reallocate, and adjust memory across multiple database buffers, shops can dramatically improve performance, especially for systems with unpredictable and variable workloads. Many modern performance solutions offer built-in intelligence enabling the system to manage itself.

The second component of database performance is the *database* itself and the structures of which it is composed. The database stores the data used by applications. When the application needs to access data, it does so through the DBMS to the database of choice. If the database isn't optimally organized or stored, the data it contains will be

inefficient and possibly difficult to access. The performance of every application that requires this data will be negatively affected.

Over time, as data is modified and updated, the DBMS may have to move it around within the database. Such activity causes the data to become fragmented and inefficiently ordered. The longer the database remains online and the more changes made to the data, the more inefficient database access can become. To overcome disorganized and fragmented databases, DBAs can run the reorganization utility to refresh the data and make the database efficient once again. But the key to successful reorganization is to reorganize only when the database requires it; instead, some companies over-reorganize by scheduling regular database reorganization jobs, whether or not the database is fragmented. Overreorganizing wastes valuable CPU cycles.

Both DB2 itself and third party tools and utilities provide autonomic methods to discover and correct poorly organized databases. Reorganization is becoming more automated with real-time accumulation of database statistics accompanied by intelligent agents that understand the meaning of those statistics and what to do when the statistics aren't optimal. Of course, the system must be aware of your unique environment; for example, you don't want an automatic reorganization to start up in the middle of your most important user's long-running query.

But reorganization is only one of many database performance tasks required. Others include data set placement, partitioning for parallel access, managing free space, and assuring optimal compression.

The third and final component of database performance focuses on the *application* itself. Indeed, as much as 80% of all database performance problems are caused by inefficient application code. The application code consists of two parts: the SQL code and the host language code in which the SQL is embedded.

SQL is simple to learn and easy to start using. But SQL tuning and optimization is an art that takes years to master. Every DBMS provides a method of inspecting the actual access paths that will be used to satisfy SQL requests. DBAs must be experts at understanding the different types of access paths, as well as which ones are best in which situation. Furthermore, DBAs must be able to interpret the output of the access path explanation produced by the DBMS, since it is often encoded and cryptic.

Host language code refers to the application programs written in C, Cobol, Java, Visual Basic, or the programming language du jour. It's quite possible to have finely tuned SQL embedded inside of inefficient host language code. And, of course, that would cause a performance problem.

Where to Start?

OK, so we've got a basic performance roadmap, but where should we start? This is always problematic. When confronted with a performance problem the best approach is to look for things that have changed recently. Changes can introduce problems that lead to performance degradation.

Of course, you might not be able to discover what has changed recently – or perhaps nothing has changed. So where should we start?

Following the old 80-20 rule, we should focus our attention on the areas that are most likely to be the cause of the problem. And, as mentioned earlier in this article, inefficient SQL is the single most prevalent cause of poor application performance. So, without any clues where to start, start with the SQL.

Finding the SQL statements that are the most expensive in a large shop can be an extremely difficult thing to do. Resource hogging SQL statements might be hiding in one of hundreds or even thousands of programs. Interactive users who produce dynamic, ad hoc SQL statements might physically reside anywhere, and any single one person who is generating ad hoc queries can severely effect overall production performance.

A good approach is to use an SQL monitor that can identify all SQL running anywhere in your environment. Typically, these tools can rank SQL statements based on the amount of resources being consumed and track the statement back to

who issued it and from what program it was called. Once you have identified the top resource consuming statements, you can concentrate your tuning efforts on the most costly statements.

However, it is not always obvious how to tune poorly coded SQL statements to make them better. So we need to go one step further by using an SQL analysis tool. Such a tool can be used to identify and explain how the SQL is currently being run and to provide a set of expert tuning recommendations on how to fix each inefficient SQL statement.

Summary

The purpose of this article is not to introduce specific tuning techniques or scripts, but to provide an overall perspective on database performance. Database performance management is a rich area for further investigation and study. Use this article as a starting point – or as a refresher if you are a long-time practitioner.

In terms of advice, the best I can offer is to be prepared. Read, study, and continue to learn about DB2, databases, SQL, and performance tuning. There is always something new you can learn, or at least something that needs to be brushed up on.

And finally, be advised that wise organizations will implement a comprehensive performance monitoring, tuning, and management environment consisting of policies, procedures, and integrated performance management tools and utilities. Failure to do so will surely result in poor database application performance.

From *IDUG Solutions Journal*, Spring 2007.

Home.