SOLUTIONS JOURNAL

The *IDUG Solutions Journal*
August 1997 - Volume 4, Number 3

Models for Estimating Performance

by Richard Bolesta & Craig S. Mullins

Most organizations monitor and tune the performance of DB2-based applications. But the performance management steps are almost always reactive. A user calls with a response time problem. A tablespace maxes out on extents. The batch window extends into the day. You've been there.

Even the proactive steps taken against completed applications in production might be considered reactive. The application is written and most changes will require some code to be re-written. Event-driven tools exist on the market that can make performance management easier by automatically taking pre-defined actions when pre-specified alerts are triggered. But even these tools must operate against a completed and running application.

A critical additional step needs to be taken: planning for the performance of an application before it is completed. This article will discuss methods to enable proactive performance engineering as applications are being developed.

Defining DB2 performance

You must have a firm definition of DB2 performance before you can learn ways to plan for efficiency. Think, for a moment, of DB2 performance using the familiar concepts of supply and demand. Users demand information from DB2. DB2 supplies information to those requesting it. The rate at which DB2 supplies the demand for information can be termed DB2 performance.

Five factors influence DB2's performance: workload, throughput, resources, optimization, and contention.

The **workload** that is requested of DB2 defines the demand. It is a combination of online transactions, batch jobs, 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 DB2 performance.

Throughput defines the overall capability of the computer to process data. It is a composite of I/O speed, CPU speed, and the efficiency of the operating system.

The hardware and software tools at the disposal of the system are known as the **resources** of the system. Examples include memory (such as that allocated to bufferpools or address spaces), DASD, cache controllers, and microcode.

The fourth defining element of DB2 performance is **optimization**. All types of systems can be optimized, but DB2 is unique in that optimization is primarily accomplished internal to DB2.

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 data page). As contention increases, throughput decreases.

DB2 performance can be defined as the optimization of resource use to increase throughput and minimize contention, enabling the largest possible workload to be processed.

In addition, DB2 applications regularly communicate with other MVS sub-systems, which must also be factored into performance planning. Many factors influence not only the performance of DB2, but also the performance of these other MVS subsystems.

Proactive performance engineering

Although reactive performance management will always be required (since systems and applications change over time), proactive performance engineering can be used to minimize reactive monitoring and tuning. Proactive performance engineering is a methodology for constructing performance models; it is done before code or database construction, and focuses on an application's run and response times. It provides a rigorous process that focuses on quantifiable results and helps to eliminate re-designing, re-coding, and retrofitting during implementation for performance requirements. Because proactive performance engineering occurs before implementation, multiple scenarios need not be coded to determine which meets the performance requirements. Figure 1 shows the relationship between the phase in which a problem is detected in an application and the cost to fix the application or database.

A performance model of an application is an analytical representation of the application's resource consumption. The model is used to estimate workloads generated by the proposed design. The resource consumption is broken down by CPU requirements, DASD requirements, and data transfer volume. The results are disbursed across a time

frame and then applied to the projected hardware platform.

Performance models will analyze the effect of all the SQL in a system based on an estimated arrival rate in terms of hardware capacity. (Most industry experts agree that approximately 80 percent of performance problems originate in the application code and database design.) Once the estimated workload has been determined, it is applied to the projected hardware configuration through queuing theory.

Proactive performance engineering is different from analyzing and optimizing single queries. Since individual queries may optimize at the expense of other queries, every query cannot be optimized. In turn, a model will show the overall effect of all the queries and how they affect each other's performance. The model allows you to optimize for overall performance objectives.

Developing a performance model is an iterative process. Designs and information supplied by various groups must be reviewed and updated as changes are made. To proactively build a performance model, management must take an active role in bringing together DBAs, application designers, and capacity planners to share information and address any business requirement issues that may affect the performance criteria.

Regardless of the project development stage, the following methodology can be used as a simple framework for developing a performance model. The description that follows is necessarily brief, given space constraints. A complete description of the Proactive Performance Engineering framework can be downloaded from PLATINUM technology's web site at http://www.platinum.com/products/ppewhite.htm.

The process of building a performance model has five phases: planning, information gathering, model construction, model validation, and model analysis.

Planning

The planning phase is the most important to the success of a modeling project. During this phase you will set the scope and performance objectives and establish communication channels between all participating parties. The planning steps include:

- Setting performance objectives -- specific and concise objectives are required to avoid confusion and scope creep when the model is analyzed. Two examples of performance objectives are: two-second response time for adding an order, and completion of a batch application in under five hours.
- Establishing communication channels -- all parties involved should be brought together and their responsibilities for the project should be stated clearly to the team.
- Setting the scope of the modeling project -- to limit the effort to a reasonable amount of work and indicate the level of detail for the model and the parts of the design that will be modeled.

- Determining business priorities -- to help focus on the components that should be modeled.

Information gathering

Information requirements for a performance model can be broken down into three categories: process, data, and hardware. Business factors can be incorporated into both the process and data categories.

- Business metrics is the information that gives you insight into the processing requirements of the application.
- Database information is the metadata of your database. In this step you must identify the required tables for the model (possibly obtaining the information from the DB2 Catalog if the design has been implemented) and determine the tables sizes based upon business factors.
- Process information describes what the application will do. The information includes the functions (programs, plans, packages, DBRMs (Database Request Modules), and the like), an estimate for the frequency of execution for each function, a determination of the unique actions to be taken against the database (i. e., the SQL), and an estimation of the access path for each.
- Hardware information includes the type and class of mainframe being used and the devices attached to it (DASD, for example). The constraints and performance parameters for each piece of hardware need to be factored into the plan.

Model construction

A model can be constructed in several ways. You can use a method as simple as paper and pencil (for only the most simple models) or a sophisticated modeling tool. Spreadsheets can be used for small models; modeling tools should be used for everything else. The building of a model will now be described, without regard to implementation technology.

The model must assign the tables and indexes to DASD devices. As multiple tables are assigned to the same device, the model will use the I/O volumes to calculate service and wait times.

The next phase is calculation. Steps include:

- Calculate the execution frequency using process information -- for example, an order entry system may process 2,000 orders a day, with each order containing 25 line items. The execution frequency of the process that validates line items is 50,000 (2,000 x 25).
- Use algorithms to calculate resource requirements -- once the execution frequency has been calculated, the resource requirements for each action must be calculated. This can be based on rules of thumb (ROTs) in absence of a

modeling tool, but ROTs must be carefully applied or an inaccurate model may result.
- Calculate total workload -- multiply resource requirements for each action by its execution frequency to determine the total workload of the model.
- Calculate service and wait times for the workload -- by applying the workload to the hardware configuration, service times can be calculated. Queuing algorithms must be used to calculate wait times. The sum of service and wait times will provide response and run times.

Model validation

The model should be validated before moving on to the analysis phase. There is no reason to spend time analyzing a model that does not reflect the current designs or one that will not meet the objectives. The two major steps for model validation are:

- Review the model
- Check for order of magnitude errors.

Model analysis

Model analysis is the process of reviewing the results of the calculation, comparing it to the objectives, and gaining insight into what drives performance of the design. First, compare the results against the performance objectives. If you meet the objectives, you will still want to continue the analysis and also build what-if scenarios.

Simple analysis can be accomplished quickly. Determine where the greatest resource consumption is in the system. Consider tuning strategies based on the results of the performance model and the information gathered as input to the model.

Steps that can result from model analysis include:

- Load balancing of DASD devices
- Modification to database design
- Tweaking of application design and code
- SQL re-writes
- Adding indexes
- Hardware changes such as adding memory or capacity (this result is unlikely).

Finally, once you have completed an initial analysis, you should build several what-if scenarios. These scenarios provide insight into how the application will handle changes. The following are just a few examples of what-if options:

- Increase table sizes
- Increase transaction rates

- Add additional indexes
- Increase execution frequencies.

Successfully using performance models

The key to using performance models is understanding that performance models are based on averages and therefore require multiple scenarios. After a model has been completed, compare the results of a performance model with the implemented system. Determine the differences and adjust the model. You can also tune a model so the next model will be more accurate. Finally, performance models are not a panacea; if a system is over capacity due to large transaction volumes, proactive performance engineering will not be your silver bullet for solving the capacity problem. It can, however, alert you to the situation before the system is implemented.

Summary

Constructing a performance model is different from proactive performance engineering. Proactive performance engineering addresses performance requirements during the design stage without losing focus on the business requirements. It incorporates business requirements and rules; this makes the modeling effort cost effective. The results will only inspire management to carry out bigger and better modeling projects.

Richard J. Bolesta is a manager of product development at PLATINUM technology, inc. with responsibility for the development of performance estimation technology. With more than 14 years experience in the information technology industry, Bolesta has been a speaker at IDUG conferences as well as at the Computer Measurement Group, Enterprise Expo, and local user groups.

Craig S. Mullins is vice president of marketing and operations for the database tools division of PLATINUM technology, inc. He is also the author of the popular book, DB2 Developer's Guide, which will soon be available in its third edition; the book will include tips, techniques, and guidelines for DB2 through Version 5.

International DB2 Users Group
401 N. Michigan Ave.
Chicago, IL 60611
(312) 644-6610