

This article appeared in the
February/March 2004 issue of



Subscribe instantly at
www.zjournal.com

- Free in the U.S. and Canada
- \$36 per year outside of the U.S. and Canada



Most performance tuning activities involve systems tuning such as managing I/O, adjusting parameters, moving workloads, etc. Only rarely do you work directly with application developers to improve code efficiency. But there's a third tuning method—one that's only rarely employed, but often can be highly effective. This is particularly true when you can't touch the application.

End-user tuning, where you work with users to improve their use of a specific application or function, can generate more balanced system resource consumption and consistent response times. Here, we'll cover examples and approaches to working with end users.

End-User Performance Tuning

BY DENISE KALM

Current Process

Systems performance management can be a frustrating job. If response time is good, you get no credit. If it degrades, you aren't doing your job. But in most companies, the performance analyst has limited knobs to tweak to improve performance. System parameters can be adjusted, memory can be used to avoid I/O, but often, the desired speed can't be achieved due to the application design.

Bad application design can limit the effective performance

of even the most highly tuned system, but rarely, if ever, is the performance analyst included on the design team. This is the recommended approach, but it occurs rarely, particularly given the trend toward purchased software. Here, we assume it's unlikely to occur.

In addition, the trend toward high level or object-oriented languages makes it more difficult for even an experienced analyst to understand or affect the impact of programming choices. System-managed storage (SMS) file management, in effect, got rid of one of the most valuable tuning techniques—hand placement of data files to optimize I/O performance. In the CICS and VSAM world particularly, this method plus buffering often resulted in 90 percent of the performance gains possible. The “guaranteed space” option can be used to override SMS file placement, but the data management group rarely supports this option, despite its proven efficacy.

A Better Way

A better way to manage performance is to begin with the application design team. Performance analysts can significantly improve the performance and scalability of an application if they're involved upfront, when the application is conceived. Management typically assumes that “performance is everyone's job,” so application designers should care about it just as much as the performance analyst. However, only the analyst is really held accountable for response time. Programmers will often cave in to users' demands, resulting in online transaction processing (OLTP) applications with batch-like or I/O-pig transactions that will never meet service level agreements (SLAs). SLAs tend to be numbers users pluck from the air, but they gain substance and reality the longer people hesitate to challenge them. Remember, an application programmer's primary responsibility is to meet the functional requirements of the business he supports; performance is a secondary consideration, at best.

Involvement in design and testing means that poorly designed transactions can be fixed while the team is still open to change. An installed application frequently attains the inertia of an aircraft carrier and, given the difficulty in passing through installation hoops at most shops, it's somewhat understandable that changes after installation seem threatening.

Unfortunately, many shops have resorted to purchased code to speed “time-

to-market.” Rarely, if ever, do the vendors support a performance or scalability clause in their contracts, and with object code only (OCO) as a standard delivery method, the performance analyst has almost no opportunity to fix the design.

Uncharted Ground

What's left? Typically, performance analysts have no contact with users and often don't even understand the business function the application supports. Why is this important? What the user needs and expects from the system and how he interacts with it greatly influences his assessment of performance. A response time number is mostly meaningless if it doesn't match the user's expectations and needs. Graphs and statistics won't help you much when a user, wearing an accountant's green eyeshade, shows you how your sub-two-second response time has cost them \$50,000 per month.

People expect certain kinds of interactions to be quick and others to be slow. They'd like instantaneous response time on all transactions, but don't expect it. For example, transactions that represent the last step of a transaction are often perceived to be complex; user tolerance for less responsive behavior is increased. Similarly, they expect quick response on a transaction that, to them, doesn't seem to require much work from the system. An example might be the expectation that a short e-mail should display instantly, whereas downloading a work of art onto their PC will take more time. It's vital to understand not what the user wants as much as to know what they expect—unless you can really dazzle them with tuning.

Understanding the business gives the analyst a huge edge both in tuning and in forecasting. During the interest rate drops in the late '80s, one could anticipate huge increases in consumer and real estate loan application volumes. But if you didn't know which systems handled the loan application process, nor which key transactions would be affected, this information was worthless. In most shops, systems programmers and performance analysts, to their peril, almost pride themselves on not understanding the business. A clear knowledge of the explicit business function of an application can help you pre-tune and get additional capacity, where needed. In this line of work, it's much easier to be proactive than reactive (and more fun, too). With time to analyze and model various solutions, the pressure's off. Particularly in the

Web space, you have a great opportunity to be a hero.

Moreover, it's vital to watch how a user interacts with the application in production. Only then can the analyst understand the limitations in translating a business need into code. We all speak English, but not the same language. What was requested (the business design) and what results (the systems design) are rarely exactly the same. Where possible, the user will frequently change his interaction with the system to obtain the desired result. This can result in performance problems—the kind you can't anticipate.

Part of interacting with the application is how a user responds to system errors or problems. Ideally, they contact their help desk and report each occurrence. In reality, users often don't feel this is productive. They develop clever workarounds, many of which are counterproductive. As the case studies will show, these kludges can lead to performance problems. Analysts unaware of this behavior may try a less-than-optimal tuning approach to address the problems. Many users believe that technical support isn't doing their job even when they fail to report the problem. Failure to monitor the user at his workstation means problems and opportunities get missed.

Another opportunity occurs with user-generated code. Some application vendors supply a workflow generator product and market it directly to users. The goal is empowerment, but the result can be poor performance. Vendors understand the frustration businesses have with their data processing departments and supply these kinds of systems as an end-run around the glass house. Often, the code is installed in production with only limited testing and no stress testing; the results can be a surprise even to the developers. If the performance analyst has developed a relationship with the user, they're more likely to be offered the option of working with workflow developers to optimize the system.

Knowing your user and his business is second only to application design in importance when managing performance. And, with a little effort, it's probably more likely to occur.

To summarize, the benefits of a good relationship with the user are:

- Being able to understand and manage response time expectations
- Obtaining information on hourly, daily, weekly, monthly, yearly business

- cycles and planned business growth
- Knowing when changes in the business will affect application use
 - Detecting unreported systems problems
 - Working with users and developers to improve their code
 - Making users aware of a performance analyst's role in supporting the business and helping them recognize the value of working with you.

Case Studies

The best way to see how user tuning works and its benefits is to look at a few examples. Most are from CICS/VSAM applications running in their own regions. Some examples are old, but still reflect the principle of working with the user to tune the system. The transaction names and details have been changed to challenge the reader to guess what the real systems were called.

The Mid-Morning Response Time Spike

A new, vendor-written consumer loan application was installed into its own production CICS region. Careful stress testing in development had revealed no particular anomalies, but almost immediately, serious response time problems surfaced. These were mostly confined to the 10 a.m. window. The rest of the day more closely mirrored our benchmark, but what was happening at 10 a.m.?

Analysis of transaction data indicated that one transaction, OLPR, was the predominant transaction during that hour and was also responsible for all response time outliers. In fact, this transaction rarely executed beyond 10 a.m., except when there were so many that they spilled past 11 a.m.

What was the transaction doing? None of the statistics were helpful and the application programmers knew only that it did some kind of online printing. Documentation from the vendor wasn't helpful, either. Unlike many print transaction designs, OLPR was not a long-running, low priority background task, reading print requests from a temporary storage queue (TSQ).

It might have been possible to try to reduce I/O with buffering, or increase the priority of the transaction to get it out of there quicker. But without knowing how important the transaction was to users, relative to all others, there was a risk that "tuning" could have worsened performance overall.

The obvious strategy was to ask the user what he was doing, but no one does that. "Talk to the user? Are you crazy?"

This approach was viewed as a serious breach in glass house protocol. But since the job involves solving problems and no one had a better idea, the next step was talking with the dreaded user. This was a good move.

It turned out that someone from the application or vendor team had mentioned in passing that online print under CICS was a "bad thing" and would hurt performance. The business logically concluded that they should minimize the impact to one hour by

doing all their print at 10 o'clock.

The management team responded favorably to a suggestion that they might try printing loan documents whenever they needed them instead of bundling them. This was actually the preferred method, so they were happy to test the theory that this would work better. The result was more consistent response time and resource use throughout the day. A side benefit was reduced queuing on the printer, a problem that had not been previously reported.



**Subscriptions to z/Journal
are *FREE* in the U.S.
and Canada.**

**Subscriptions outside the
U.S. and Canada are
\$36 per year.**

[Click here to subscribe.](#)

Virtual Storage “Creep”

There may be many “creeps” in the systems world, but the one that used to be most dreaded was the entity that would start eating its way through the below-the-16MB line virtual storage, a limited and valuable resource.

Back in the days when most programs ran below-the-line, and CICS systems programmers actually did capacity planning on virtual storage, the size of program libraries was of great concern. After running reasonably well in production, another loan application writ-

A response time
number is mostly
meaningless to
users if it
doesn't match
their expectations
and needs.

ten in COBOL began to run short on storage. The problem increased over time and the “system under stress” message began to etch itself on my monitoring screen. Monitoring indicated that the biggest dynamic storage area (DSA) user was program storage.

The application developers were stumped. Though this had originally been a vendor package, they now controlled the code and they hadn't changed

anything in quite a while. The number of users hadn't increased; the transaction volumes were reasonably stable; and there was no indication that users had changed their work patterns to concurrently exploit more objects. A quick scan of the program processing table (PPT) indicated that no additional programs had been marked resident.

The user wanted a meeting with the application and support teams as soon as possible to hear how we were going to solve their problem. We were mostly sure nothing had changed on our side, but could something have changed on the user side?

“We haven't changed anything,” the loan department manager began, “Except for tailoring a few screens.”

Actually, more than a few. This vendor application let the user create and tailor screens in real-time without going through the laborious, time-consuming change control process. What no one knew was that each new screen and each tailored version of a screen became a new program. The old templates remained intact for others to use. As every user had this ability and no one knew the cost, almost every loan officer had created his or her own personal screen library.

Working together, we arrived at a business-focused solution, designating a small group of users as the focal point for all changes. They helped identify and delete minimally used screen designs, combined the wish lists of many users into a smaller set of screens and otherwise substantially reduced the run-time library size. Limited storage conditions became history.

We could have insisted on taking over all program management, but by teaming with users, they became real partners with us—as responsible for system availability as we were.

User Tuning Skills

So what's required for successful user tuning? What skills are needed, and, as a systems programmer, what changes might you need to make?

First, remember that you're not here to fix the user. The problem is with the system and its failure to meet the user's needs. Partnering with the user may also result in changes you and the systems group need to make in your way of thinking and your actions and interactions with the user. “People” issues are rarely straightforward or easy to define; take this opportunity to work on them anyway and improve your:

- **Empathy and respect:** You should exhibit a desire to obtain the kind of performance improvements most visible and desirable to the user. Many performance analysts feel beholden only to their management and to metrics these managers understand. Improving the performance of the transactions the users care about most may or may not translate to improved metrics. You may be charged with maintaining average internal response time at 0.5 seconds, but all the user cares about is a single transaction—one that generally averages well above your management metric. Tune to their needs, not a global average. Another disconnect comes in defining performance. Examples of user performance metrics might be the number of authorizations processed in an hour, or the cost of processing a check. The more you can translate between system metrics and business metrics, the more satisfied your customer will be. Respect their knowledge and they will come to respect yours.
- **Business knowledge:** It'd be nice if your users could translate their business-speak into technical requirements for you, but frequently, they can't. Often, business terminology is no easier to translate than technical terms. Knowledge of your industry and the ability to flesh the picture out with their business drivers and priorities will help you expedite the process. Ask them about their business cycles. You might be surprised to know that they can tell you peak hour, day, week and month, without ever running a report on the system. The ebb and flow of business volumes can turn the performance job from problem management to proactive tuning.
- **Communications skills:** You may be a terrific presenter to the technical community, but how well do you communicate with your teenager? Are your message always get through? For user tuning to work, your ability to clarify the complex is essential. The people you'll be working with are sharp; they didn't get where they are without talent and ability. They just have their own language and jargon. A good sense of humor, translation skills, awareness of history, and a solid vocabulary are the tuning tools you need.

Here's a typical systems programmer and user conversation:

“First, we want to port the application to client/server, probably AIX on SP2

with an Oracle DBMS. Then, we'll GUIfy your front-end."

User looks suspiciously at her chest. "You want to do what to my front-end?"

A better way to conduct the same conversation might be:

"You've told me that your data center charges are a real concern. Here's a spreadsheet demonstrating how we can reduce your unit cost by moving your application off the mainframe, onto smaller, cheaper processors. You've also said that you don't like the "green screen" look, since you paid for high-end PCs. You want color and a Windows-like screen; we can do that for you, too."

Additional traits you should work on include:

- **Effective listening and patience:** This process is new to users, too, and could be initially perceived as threatening. Think of it as a new way to increase your business knowledge.
- **Tact and diplomacy:** Many user groups treat the technology organization as the enemy (and vice versa). Given that you want to analyze their processes and suggest changes, it's vital to understand that this might not be interpreted as beneficial unless you can successfully sell the idea. Be upfront about how the system hasn't met all their needs (unless they selected it themselves), and offer suggestions that make the best of things. To many, the system is just a recalcitrant, troublesome tool they use to get their business done. Few will be impressed by the technology as such.
- **Detective instincts:** Most opportunities aren't immediately obvious and, sometimes, you can almost tune around the real causes of the performance problem. This isn't ideal and usually costs significant resources. Check all assumptions. As in the previous examples, if it looks like something in the system has changed and caused a problem, and you know you didn't change anything, then you have to look at what the user could do, even when you're absolutely sure no one outside your department has the ability to change the application code.
- **Open mind:** Whenever you venture into the unknown, you must leave your expectations and assumptions behind. This is an excellent learning opportunity and the results can be exciting. But it isn't the way you were trained to work and changing your behavior may initially feel strange.
- **Risk-taker:** Try to be a pioneer and embrace the chance to work with a

user. You may not be entirely successful with every line of business or person you encounter. You'll have to try new approaches and learn a different mindset. But, ultimately, you'll make a positive contribution to the bottom line, turning yourself from a cost center into part of the profit-making machine.

Once you feel confident with these skills, you need access to the user. This one is trickier. Systems programmers and performance analysts typically don't know the user. Cultivate relationships with application developers, user liaisons, etc., to get to the right people. In many shops, there's a virtual iron curtain between users and systems people. Tread carefully and wave an olive branch.

Once your mission is clear, you'll find real support. Most users really appreciate having their needs addressed. Don't stop with business managers; find an excuse to go out on the floor and talk with those who live and die by the performance their system offers. You will quickly learn how a relatively minor problem in your view can interfere with the real work of solving customers' problems.

Remember the 90:10 rule, which says that most of the gains happen with small changes in the way everyone does things. So, don't overture. And if the user doesn't like the change and you can't persuade or negotiate your way to success, realize that you can't win by escalating.

Summary

End-user tuning is an area of systems performance rarely pursued, but sometimes, it's the only way to meet your SLAs and satisfy your customer. Too often, we forget why we're here and what the application is really doing. A relationship with the user and a business view of the application will help you add valuable performance tools so you can maintain acceptable response times at acceptable cost. **Z**

About the Author

Denise Kalm has more than 20 years of experience in IT, most of that time in the performance management/capacity planning arena. Starting out on Tandems, she branched out to CICS, MVS, Unix, and began looking at network performance for a large, global bank before joining BMC Software, Inc. as a software consultant. Her current role is in technical marketing for performance products.
e-Mail: Denise_Kalm@bmc.com
Website: www.bmc.com



**Subscriptions to
z/Journal are FREE in
the U.S. and Canada.**

**Subscriptions outside
the U.S. and Canada are
\$36 per year.**

[Click here to subscribe.](#)