It's Time to Automate Your Application Performance Monitoring Process

BY MOIRA MCGLYNN

If performance monitoring tools are put in place early in the application development lifecycle, widespread Y2K and Euro currency conversion updates can be tested for performance concerns. These tools will help integrate monitoring in the application lifecycle and assist in assessing the magnitude of performance problems. Then, the decision can be made to tune now or wait for a more convenient time.

IT'S a little over a year until January 1, 2000, and you're in the midst of Y2K updates and possibly Euro currency conversions. You may be outsourcing these rewrites either in the United States or on foreign soil. In any case, you're carefully tracking costs as well as the all-important timetable to avoid any possible Y2K litigation after you enter the next millennium. But what about all those code changes to your application portfolio and their effects on the application's performance and on performance of the system as a whole?

Take, for instance, the typical mainframe environment. Systems people focus their efforts on monitoring performance of the overall system. They often find it difficult to schedule in-depth performance monitoring of every possible problem application, let alone all the applications in the company's portfolio. Regrettably, many application performance problems surface in production, when costs to fix are high and ramifications of performance degradation are the greatest. This is due in part because application performance measurement involves systems skills and not the application design and programming skills developers wish to hone; and in part because application performance monitoring is tedious, time consuming and difficult to do in the controlled test environment. For many reasons, application performance is one aspect of software quality often not fully addressed.

So, today, while you are undertaking the task of widespread Y2K and Euro code redesigns and rewrites, consider the performance of your altered applications. By automating tasks involved, an application performance monitoring process is more easily implemented early in the software's development when it is easier and less expensive to tune code. Automation may be an easy, efficient and cost-effective way to integrate a performance monitoring process into the existing application lifecycle and software development "culture."

WHY PERFORMANCE TUNE APPLICATIONS WHEN WE ALREADY TUNE SYSTEMS?

Most shops that own their own MVS environments usually performance monitor and tune their system or subsystems. Some may do this out of necessity, because users demand a certain response time level or specific jobs must complete execution before others running at the same time. Other MVS shops may strive for a finely-tuned system and routinely optimize existing capacity to the highest degree. Most companies are somewhere between the two, providing at least a minimum level of performance and improving upon it whenever possible. In any case, someone usually monitors MVS system-level performance on an on-going basis.

In contrast, application performance in many MVS shops is often handled reactively and selectively. When one application eventually "hogs" enough production resources to be noticed, a systems programmer or performance analyst will usually focus on it, monitor its system resource usage and attempt to narrow down the problem code area. The next step is to involve a developer familiar with the application's functions to assist with developing a fix. Some companies enhance this process by periodically isolating several high resource-consuming applications for tuning. Even with this effort in place, hundreds of thousands of lines of application code not making this "high profile" list may never be performance tuned. Untuned applications are often not

the result of negligence but are casualties of a tedious, time-consuming performance monitoring process.

This distinguishes the system approach to performance concerns from the typical application approach. For example, consider a batch payroll application that runs every Thursday from 8:30 p.m. to 8:30 a.m. Friday. Twelve hours is too long of an execution window since the day shift begins at 8:30 a.m. and the users expect to sign on to CICS upon arrival. The systems programmer or performance analyst may monitor the payroll job's performance during its next execution and suggest that Payroll modify the application so it runs in a higher priority job class and calls a faster sort routine. The Payroll group makes the necessary JCL changes, and this Thursday the payroll job runs eight hours instead of twelve. The next step in the process of "trimming the resource fat" is to measure the performance of the application code in more detail to focus on lines of code that are causing the problem. The measurement information should provide in-depth information, such as the database, file or tape access method used, how and when files are accessed and the I/O or EXCP count for each, file blocksizes used, the efficiency of SQL statements accessing columns in a DB2 table, etc.

A batch window that encroaches on the online window, as the above example illustrates, is one reason to performance tune applications. Other reasons are to improve response time, to save system resource costs, and to avoid hardware upgrades. And, if you market your applications, you will want to consider your company's reputation as a resource-efficient software provider as well as save future maintenance costs. These are costs associated with isolating an application performance problem, testing a code fix and shipping the updated code to the customer base once the application becomes widely sold and used in the marketplace.

The bottom line is that system-level performance tuning may not be enough to improve response time to service level agreements or to hold off purchasing a more powerful processor. The next step is to scrutinize the applications that run on these finely-tuned systems, particularly now that a good deal of mainframe software is in the process of Y2K and Euro currency conversion rewrites.

WHY IS TODAY A KEY TIME FOR APPLICATION PERFORMANCE?

Erroneous code is often introduced in software whenever it is reworked. For this reason, developers use a quality assurance (QA) process to improve and certify application code quality when it is updated with today's Y2K and Euro conversion changes. This QA process typically includes some combination of code inspections, unit testing, function testing, regression testing, and overall performance and system testing and is used to assure functionality and drive out injected defects. With all the widespread Y2K and Euro currency conversion application code changes, a significant amount of code is either currently or will soon be in the QA process anyway. Now is the time to incorporate application performance measuring and possibly tuning into the QA process to further enhance software quality. The importance of performance measuring at this time of widespread code rewrites cannot be overly stressed. At the very least, problems will be uncovered and assessed, even if tuning is not immediately attempted due to time constraints.

When application performance problems are discovered early, the potential for savings is greatest and the impact of a performance problem smallest.

Like many companies today, yours may outsource development work, particularly Euro currency conversion. Ask yourself, "Will someone else coding my application care as much about its performance as we do?" If you can't answer "yes," you would benefit from benchmarking the application's performance before it leaves for outsourced updates and again when it returns. Performance measuring before the application is outsourced allows you to figure resource usage levels into a contract and ask for accountability after code is updated and returned. If you can't include such requirements in the contract, prior performance measuring will provide you with data on average resource usage for measuring and tuning the code upon its return.

If you market your own applications, you have additional interest in their performance. Your customers probably vary in their degree of performance focus for their own applications but you can be certain many of them will compare your application's performance to that of your competitors' when purchasing decisions or license renewals arise. Whether the code conversions are outsourced or tackled inhouse, today's massive code rewrites are prone to future problems and your customers know who performance tunes and who doesn't.

WHAT DO YOU AUTOMATE?

A useful software maintenance adage to remember is this: The more you update code through rewrites and redesigns, the more likely you are to introduce defects into the code. These defects will surely include performance problems. If you are still very early in your Y2K conversion, you have read this article with incredulity up to now. "Why would I take time to measure and tune the performance of my applications when costs are accumulating, the clock is ticking and I have no skilled personnel to spare." In this case, you will want to assess the cost of automating the performance monitoring process to some degree and weigh this cost against the benefits gained from well-tuned applications. These benefits include long-range savings over several years realized from a well-tuned application portfolio efficiently utilizing system resources. Tangible proof will be total service units saved, the batch window that met its schedule, the hardware upgrade avoided, response time improvements that increased productivity, and customer accounts gained by the company's enhanced reputation for performance efficiency. These benefits must be weighed against the cost of automating and implementing the performance monitoring process, the cost of which will depend on the scope of the work automated.

Some tasks in application performance monitoring easily lend themselves to automation. The repetitive process of measuring and recording applications' system resource usage and isolating the highest users on a regular basis are examples.

Many MVS shops already automate this preliminary step of the application performance monitoring process. MVS maintains resource usage information in its System Management Facilities (SMF) records. This SMF information ranges from basic data such as CPU time consumed or EXCP counts generated to more in-depth details

including the number of tape mounts required by the job or the amount of CPU time needed to process I/O interrupts. SMF 30, subtype 4 or "Step Total" records include data on the total resources used by a job step while subtype 5, or "Job Termination or Termination of Other Work Unit" records provide information for the entire job. Homegrown or pre-packaged software can be used to automatically read and interpret four or five basic SMF system usage data to detect either high usage applications or that occasional spike of usage above an application's norm. Companies use this filter data to focus on likely candidates for further performance monitoring and tuning. Additional automation can then check the problem application's source code for a developer's name and prepare a file to email that individual with details of the resource usage anomaly. If the developer is familiar with recent updates to that application code, he or she may quickly implement a fix.

If preliminary data does not help formulate a quick solution, more detailed application performance measuring is done and the data is reviewed by skilled personnel. Either the developer or someone familiar with that application's functions is involved at this point to help interpret the performance data, isolate to a problem line of code and attempt a fix. The application is then measured again to ensure the code update improved performance to the desired level.

Some of these more detailed measuring tasks can be automated to a degree. After automatically checking SMF information for a system resource usage anomaly on the initial run, more detailed resource information can be automatically gathered by inhouse or prepackaged measurement software. In the case of an application utilizing an unusual amount of CPU time, a bit might be set in the automation software. This bit could signal for more in-depth SMF or other resource data collection the next time that problem application executes and/or to issue a VSAM utility IDCAMS command "listcat all" to generate a detailed file information report. This performance information might reveal extra VSAM processing due to incorrect freespace allocated for the file, which could be calculated from "listcat" output.

Software development companies already employ automation tools in the development process. Many of the latest RAD tools help automate application development tasks such as design modeling, code generating and library maintaining and building. Now is the time to include performance monitoring tasks in your company's automated processes, and a cost-benefit analysis can help determine how much automation is realistic.

INTEGRATING PERFORMANCE MONITORING INTO THE APPLICATION LIFECYCLE

Ideally, the maximum benefit of automated application performance monitoring is derived from monitoring not only the most problematic applications but all performance tuning opportunities. A single CPU second saved by tuning one online transaction sounds trivial, but it isn't if that transaction runs 300 times each workday. Estimate a cost of $.12 per CPU second per month and the savings from this "trivial" performance tuning effort is more impressive. Such simple tuning opportunities may span the application portfolio, and it's worth comprehensive monitoring with an automated tool to find out.

The question is whether to face the application performance issue now with as much information as possible or wait for problems to surface adhoc, when Y2K and Euro currency conversion code is executing in full swing, possibly at hundreds of sites.

Application performance monitoring and tuning achieve the greatest savings when done early in the application lifecycle. When the application is still in the QA environment, its developer is usually available to quickly remedy a performance problem. By the time that application is executing on several internal and/or hundreds of external customer sites that developer may be assigned to new projects and unavailable to work on this one. It is less costly and more efficient to retest software still in the QA environment, when fewer tests need repeating.

While it is more difficult to do, application performance measuring can be accomplished in the test environment by utilizing automated stress or load test tools. These tools require thought in creating test "scripts" which can then be reproduced and timed to execute in a way that mimics real-time users. Guenter Priller, an independent consultant working in the area of application performance has done just that with online applications. "I usually take 15 to 30 minutes of test time, running the application with Platinum Technology's TransCentury Enterprise Tester and measuring this window with Programart's STROBE," said Priller. "This leads me then to a prognosis on the behavior of the application, which I can approximate to the production environment. Further on, this gives you the input to capacity planning." Priller continues, "For performance reasons, you can control the script flow by think times, simulating the coffee break, lazybones and tough workers. You can play real life."

Stress testing generally helps to drive out more visible defects, but is a useful automated tool to identify performance problems in application code as well. When monitored in the test environment, some of the typical "firefighting" tasks are lifted from busy systems people — they don't have to work overtime to put out a production fire created by a problem application. Automating performance monitoring with automated stress tools for use in the QA test environment would be the ultimate objective in application performance monitoring.

Performance monitoring early in the development lifecycle can more easily happen once these tasks are automated and automated stress testing is in place. When application performance problems are discovered early, the potential for savings is greatest and the impact of a performance problem smallest.

CONSIDER THE IMPACT ON YOUR CULTURE

Utilizing automated performance monitoring tools during application development need not dramatically prolong the schedule. The purpose of the automated tool may be simply to isolate problems, thus allowing time to assess their impact on performance and determine the feasibility of implementing a fix. And, not all performance problems require a code update. The performance fix can be something easily accomplished, like an increase to the number of strings for an online VSAM file or a change to file buffer sizes to enhance I/O efficiency.

With an automated monitoring process, more application performance tuning opportunities will be revealed. Additional development skills may be needed to research remedies for performance problems.

If application performance training for internal personnel is not cost or time effective, there are specialists available for consultation. Then, internal training in performance tuning would be held at a more convenient time. Once trained, developers can incorporate performance techniques in code the first time through, thereby increasing the efficiency of the performance monitoring and tuning process.

Application performance monitoring and tuning often require a team approach even with automated tools. That team consists of personnel with system knowledge to interpret system resource data, as well as application development skills to provide information on functionality and to consider improvements. Increased communication and cross-training between the systems and development groups may be required. Time freed by automating more tedious tasks in the performance monitoring process can be reinvested where it is most useful — in developing the teamwork needed to actually tune applications.

A top-down management approach is needed to focus individual efforts on performance efficiency in application development and to motivate any cross communication and training. Service level agreements and internal cost-back accounting remind development teams of the cost to execute their programs. Considering performance efficiency in salary reviews and including it as a company award category reinforce the concept that performance efficiency saves money which is then passed along to high achievers.

CONCLUSION

Automated performance tools will help integrate monitoring in the application lifecycle and assist in assessing the magnitude of performance problems. Then, the decision can be made to tune now or wait for a more convenient time.

If performance monitoring tools are put in place early in the application development lifecycle, widespread Y2K and Euro currency conversion updates can be tested for performance concerns. If not, a company runs the risk of performance problems spanning a large body of updated application code, even in the batch program's critical path or in critical online transactions, that will surface at any time. Without automated tools to test the performance of all changed applications, overall performance efficiency is an unknown. That means future response time degradation, system usage cost, batch window schedules and hardware capacity are unknown.

The question is whether to face the application performance issue now with as much information as possible or wait for problems to surface adhoc, when Y2K and Euro currency conversion code is executing in full swing, possibly at hundreds of sites. ts

Moira McGlynn currently works for Emerald Software and is involved with automated application tools. She began her career as a programmer in Technical Support working on the MVS operating system environment and later with MVS applications. She has 10 years of experience in various aspects of mainframe Technical Support. Her work in mainframe software development planning led to research for a Ph.D. in Administrative and Engineering Systems from Union College, N.Y.