

Four Steps to Faster Batch



BY WES SCOTT

Batch tuning can be an overwhelming project from the start. However, as this author demonstrates, by keeping a disciplined approach to the tuning process, you can take on the analysis of a complicated system of thousands of jobs and turn it into a very manageable endeavor.

While the technical “welcome mats” of companies have shifted increasingly to the Internet and servers, the back-office batch stream of work continues to be a vital aspect of business. To this end, the speed at which this work can be accomplished can be a strategic advantage to the firm. To accomplish a state-change in the way batch work operates requires a planned process to fully diagnosis and tune an often-diverse collection of work.

When undertaken with a disciplined adherence to “what really matters,” you can see through the enormity of the job, find the root problems and come out a winner! How? Here are four steps to successful batch tuning:

1. Fully define the problem and/or objective.
2. Establish a base model of the cycle and its critical path.
3. Find and resolve the problems on the critical path.
4. Ensure that the improvements are implemented.

WHAT IS THE OBJECTIVE?

Before you can ever hope to solve a problem, you need to fully understand what that problem is. Batch production customers are a varied lot; however, many times they share a common complaint: the job is late. What do they mean? What is considered late?

For some customers late means that a report or posting is not getting done on time. Which specific report or posting do they mean? Where does it come from? You need to know what the customer needs done and when it needs to be done.

For other customers the online processing regions that support the customer service unit are not up early enough or are not

consistently on time enough. What is keeping the online regions down? Does it need some action by the cycle or is the customer bumping up against an arbitrary administration policy? Sometimes a solution does not even require a change to the workload; sometimes it is just a perception or rules problem.

For some customers the online regions are forced down too early. Today’s e-commerce means that production facilities running in the Eastern time zone need to support an Internet inquiry from the west coast at a reasonable 9 p.m. Pacific time. Why does the cycle require the region down so early? Does the cycle run too long for it not to start that early? Could adjusting the cycle’s work schedule save an hour on the front end? Defining the customer’s needs is critical to finding an effective solution.

Finally, the customer you are trying to satisfy might be your own group. For example, if a cycle could run in a shorter time, or perhaps not shorter but simply

more efficiently, then new work could be added without hardware upgrades and/or expansion. Offsetting normal growth expense can be an objective in itself.

However, without fully understanding the problem and having an expectation of what is considered a “solution” you can never hope to succeed!

ESTABLISH A PREDICTABLE BASE CYCLE

You need to ask what tasks comprise the cycle. Many shops do not know or do not have a complete picture of their cycles. Shops that have established Service Level Agreements (SLAs) usually know what task execution times are key to making the SLA. However, what predecessor tasks need to run to accomplish this work may still be a mystery.

This key point is also why general tuning of an operating platform and/or system resource for all users or jobs rarely provides much relief for a specific objective. A rising tide lifts all ships!



FIGURE 1: A CHART OF CYCLE END-TIME HISTORY

For one thing, cycles are constantly changing. Keeping a complete cycle documented is an entire job in itself. In addition, it is not as if you can stop the changes. The work and capabilities of a cycle do not exist independently of the business; rather they are intimately related to business objectives. Consequently, as the business changes so must the work.

Therefore, a key initial activity is documenting what the cycle looks like. What tasks are included? What dependencies does one job have with another? Moreover, where does it all start?

The next step once you have an idea of what tasks and task relationships constitute the cycle is refining what the cycle looks like when the complaint is lodged. It could be that only certain instances of the cycle's execution are providing all the complaints. Maybe no one sees the late run except when the weekly master file updates occur on Fridays. On the other hand, maybe it is the month-end account-closing version of the cycle that raises the complaint.

Using the simple abstraction of cycle end times over instance history is an excellent tool for quickly finding common variations of a cycle's executions. Instances that tend to end about the same time are often related and a good potential collection of instances for a modeling base. Figure 1 shows an example end-time history report as produced by Lexonix Technologies' VisiTrac product. Note the saw-tooth pattern of end times re-occurring on a regular basis.

It is important to have a reliable and consistent base model of the cycle you are asked to improve so that you can identify the true problems from the transient and/or unrelated ones. Many scheduling products attempt to give you some insight into such an average cycle view. However, as an

Instances with common end times are usually quite similar.

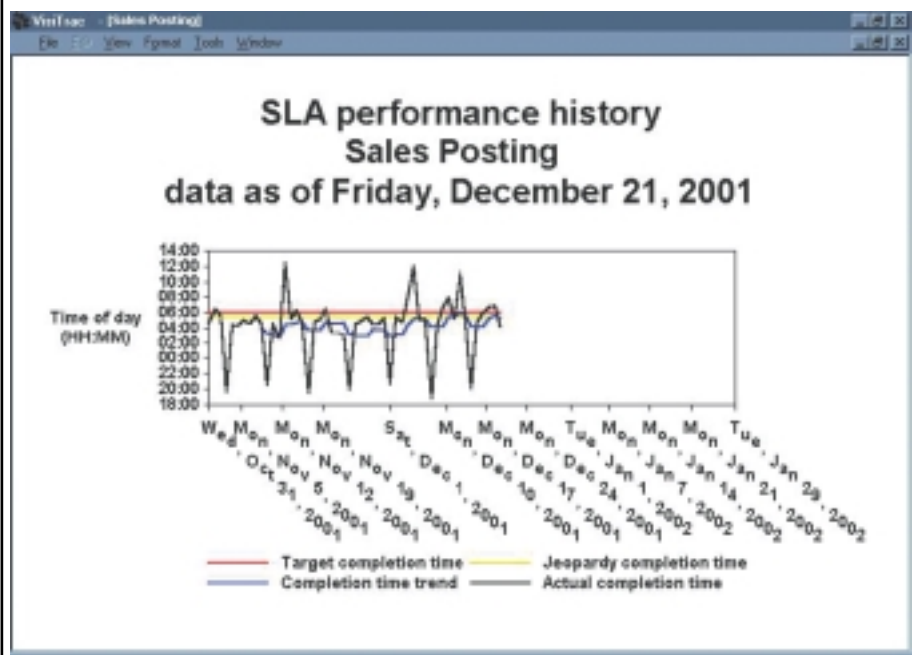
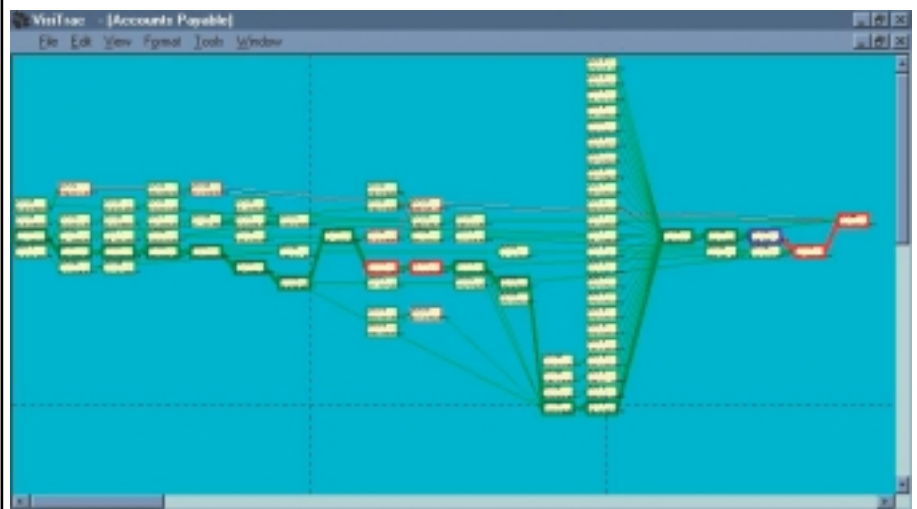


FIGURE 2: A FLOWCHART WITH DYNAMIC CRITICAL PATH RECALCULATION

This type of flowchart is perfect for modeling your improvements.



example, often their concept of forming an “average” uses the last five executions: five runs that may include three daily runs, one more active Friday run and one nearly anemic Saturday execution. You cannot rely on this average when searching for the problems and modeling solutions.

What you need is a view of the cycle where the tasks are known and the execution timings and averages for each job reflect a collection of executions from substantially similar runs. Collecting a model from a daily cycle problem is not difficult. However, what about a problem that only occurs on Fridays, month-ends or quarter-ends? The data to draw averages from is usually available; however, building a good model requires identifying and consolidating the correct observations.

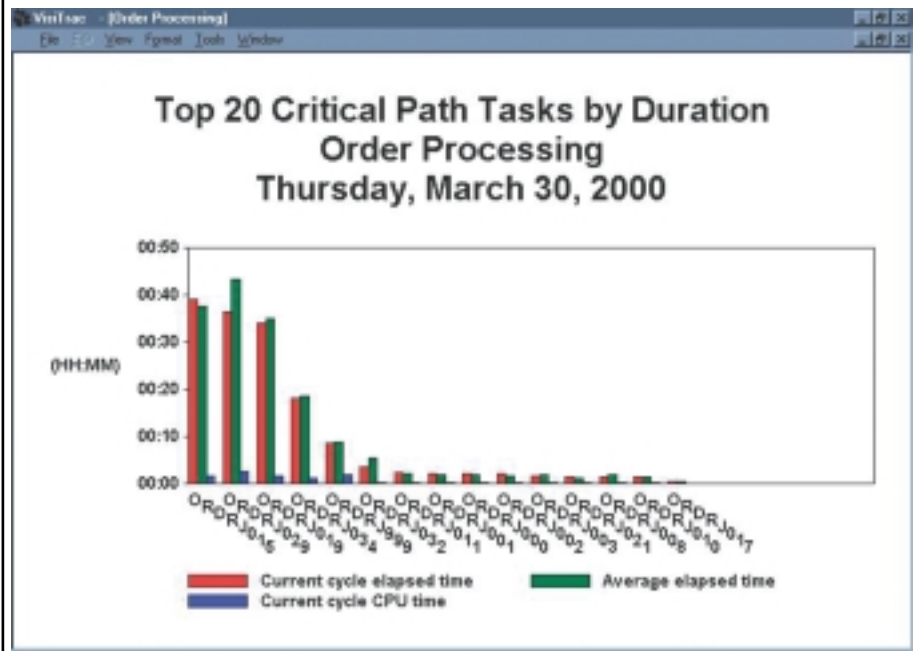
Once you have a reliable model of your cycle you are almost ready to start attacking the problem. The final activity is taking that model and identifying the critical path. The term “critical path” often leads to many mistaken identities. Many groups have a concept of critical path as being the “important jobs,” perhaps jobs that use a “limited resource” or are enshrined in company history as the “keys to the cycle.”

When you need to meet an objective, there is really only one definition of critical path that matters. A critical path is a sequential path of predecessor jobs, which lead to end points that make or break the cycle’s timeliness and SLA requirements. More specifically, if you have jobs A, B and C as predecessors to job X, the critical path from job X will lead back to the predecessor job or jobs that end the latest and thus holds out the start of job X the longest. Figure 2 provides an example batch cycle flowchart. Jobs are represented by boxes. Job dependencies are indicated by arrows that connect predecessor jobs to their successors. The critical path jobs are indicated using a bolder outline and arrow. Jobs that are red or blue indicate lateness, exceptionally long run times or suspected abend/restart sequences.

Herein lies the key to a successful batch tuning exercise: Fix the jobs in the critical path, where ever that path may go and/or change, and you will be in the money every time! Do you fix the “other” (non-critical path) jobs? Well, maybe. Certainly you do if fixing a critical path job causes the critical path to wind through the previously non-critical path job in question. You might have to fix a non-critical path job to account for a

FIGURE 3: AN ORDERED CHART OF THE 20 LONGEST RUNNING CRITICAL PATH TASKS

Often a few jobs in the critical path account for the majority of cycle time to save.



fundamental change to the overall cycle needed by a critical path job. Otherwise, leave the other jobs alone!

This key point is also why general tuning of an operating platform and/or system resource for *all* users or jobs rarely provides much relief for a specific objective. A rising tide lifts all ships! Improving performance for all jobs keeps their relative performance differences constant. To make the dramatic improvement you need to focus your improvements on the jobs that count — those on the critical path.

FIX THE PROBLEMS ON THE CRITICAL PATH

Now you are ready to locate and attack the problems that are obstructing your objective. Start at the end — the end task — and work your way backwards to each critical path predecessor.

Remember that the end job you want may not be the last cycle job to finish. Go back to your problem definition. The “end job” you want is the specific job or jobs that are closest to the problem you are addressing. For cycles that are delaying a follow-on activity, the “end job” is a pretty straightforward selection. For cycles that must start later to avoid overlapping a pre-cycle activity, you probably need to work from the latest ending end job and work all the

way back to the cycle’s start. To accomplish this, it is handy if the flowchart tool lets you select jobs as critical path terminus jobs even if they are not the whole cycle’s latest ending job.

Sometimes end job(s) are just “rallying” or synchronization points. They often serve no purpose other than to act as a successor to all the major streams of a cycle. Thus, when all the predecessor streams have completed, this task runs and marks the cycle as being complete. Although you probably cannot do much with this job, it provides a good starting point.

Start at the last critical path job, see if improvements are possible, apply an estimated improvement to the task’s duration, and then look for the next job along the critical path that has “potential.” In its simplest form, jobs with potential impact are those that have an overall elapsed time significant enough that saving 5 to 30 percent of the task’s overall execution time can effect the cycle’s duration and/or alter the critical path by demoting the importance of that job to accomplishing the objective.

At this point, it is interesting to note that while looking at critical path jobs, only a few of the jobs typically represent the majority of the critical path’s duration. However, this does not mean that once these jobs are fixed you will be done. You will need to look at many more jobs. Sometimes

the biggest time savings will come from totally unexpected places. The bar chart in Figure 3 shows the longest running (up to) 20 jobs in the critical path. Note that only a few jobs constitute the majority of time. Viewing your critical path jobs in this manner simplifies the identification of candidates for tuning. Another advantage of such a report is for monitoring the health of your cycle into the future. This chart shows current job duration against its average. After tuning is done, a future late cycle is likely to be the result of a current critical path job duration greatly exceeding its average; such a report makes it easy to find the culprit.

With a target in your cross hairs, it is time to analyze the task's execution and look for opportunities. A convenient approach is to determine the average duration of each step or phase of the job or process — assuming it is a multi-step item. Here again, look for the steps that have some meat in their execution duration. Then, drill into those steps and see what makes them tick.

The best way to do this is to identify the major components that make the step run as long as it does. Typically, these components are broken into synchronous and asynchronous items.

Synchronous items, as the name implies, means items that tend to occur in a serial fashion and prevent other activities within the step from proceeding until they complete. These include the following:

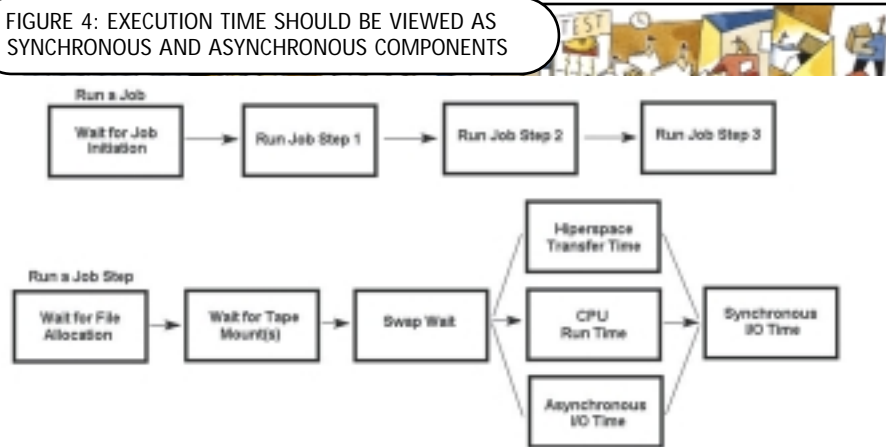
- time waiting for:
 - step initiation or initiators
 - file allocation or recall
 - tape drive allocation and/or mounts
 - a logical swap and/or virtual memory paging
- time executing on the processor
- time executing serial, sequential and/or non-cacheable I/O

Asynchronous activities are usually I/O to a cacheable, asynchronous or data base control region. Figure 4 provides a conceptualized view of these components.

Here are some improvements to consider:

- Buffering for high-use files: There is no better candidate for memory use than highly used files in the critical path.
- Using good block sizes for files based on hardware geometry or allowing SMS-enforced block sizes. Old block size

FIGURE 4: EXECUTION TIME SHOULD BE VIEWED AS SYNCHRONOUS AND ASYNCHRONOUS COMPONENTS



definitions tend to get “grand fathered” in while hardware and access methods change over time. Be sure your definitions are current.

- Moving small tape files to disk to reduce setup time and enhance parallelism. While avoiding disk usage may have made sense long ago, today's storage costs may have changed that paradigm.
- Avoiding allocation delays caused by tape mounts, HSM recalls or access contention with other jobs. In OS/390, any delays in allocating files occur after the initiator has been assigned. While the system waits for these allocations to complete the initiator is not available to start any other work.
- Use proper priority, performance groups and policies for your critical path jobs. Critical path jobs should always get the best access to resources available.
- Provide premium service to sorts in the critical path. This is the best place to apply the fast sorting technology.
- Unless the sort result is used by other tasks, make the sort part of the step that needs it. This saves Close/Open processing time.
- Investigate Data-in-Memory techniques for critical path files to avoid I/O. When multiple critical path jobs require data access, techniques such as VIO, HIPERBATCH and BLSR are great ways to increase access speed and parallelism.
- If a large portion of the step time is in CPU usage, maybe some code improvement is needed.

Typically, the synchronous components can yield the most relief. However, do not

forget to check out any substantial asynchronous activities. Poor cache support, insufficient buffering and storage subsystem constraints on asynchronous data access can still add up to quite a bit of step duration and should not be ignored.

Once you have determined the potential savings for this job, estimate a revised duration, enter that new duration into your model and see where the critical path leads you.

Did you catch that? Keep track of the effect your savings will have on the interconnection of job dependencies and the resulting path of the critical path. You should use a tool that projects timing changes in predecessor jobs through the successor chain to all down-stream jobs. If you have saved time, that time should be reflected by all dependent successors now starting earlier. In addition, that cascading effect of time changes from the point of savings to the end of the chain may very well move the critical path somewhere entirely different in the collection of tasks that make up the cycle.

Do not be discouraged! You may find that you will need to work on a job that runs much later in the cycle than the one you have just fixed. That is OK. The important thing is that job is where you need to work next. Remember, always follow the critical path!

Keep working the critical path until you reach the “start” end of the cycle. Did you reach your objective? Has the overall cycle run-time decreased so you can either start the cycle later or expect it to finish sooner? If so, you are done analyzing. If not, then it is time to refocus your view. You need to look for opportunities on a broader and subtler level. Here are some ideas:

- Remove unnecessary job dependencies from the schedule.
- Use step-end dependencies instead of job-end dependencies if the true dependency is resolved early in a predecessor's multiple steps.
- It may be convenient for you to divide and run as parallel jobs steps that have been bundled together. Sometimes, in multi-step jobs, the steps in the front-end and back-end have no relationship to each other but were placed together because it was convenient. If this is the critical path job, the steps should be divided into separate jobs for greater parallel execution possibilities.
- Check if backups have been included in the critical path. If they do not need to be part of the critical path, remove them. If they do need to be, such as point-in-time image backups, consider a concurrent or "snap" copy technique.
- Move your most commonly used critical path files to better performing hardware. If the files are tape-bound, moving them to disk may allow jobs to run in parallel instead of serially. Although this may only save seconds per job, it is possible to achieve savings of cycle-wide minutes.
- Look for underlying constraints in your storage subsystem. High-use files can cause a lot of access serialization. For example, the Index component of one VSAM file placed on the same disk volume as the Data component of another VSAM file can cause both files to perform poorly. Move high-use files to separate volumes or subsystems or use a virtual allocation Unit Control Block (UCB) approach (such as PAV) to avoid reserve/release serialization.
- If you are using a "hyper volume" style subsystem, check if you are actually allocating high-use volumes to some common hardware back-plane element and eliminate this coupling if found. Even with today's large subsystem caches, major sequential operations can quickly overwhelm cache and restrict the file to the back-plane access speed.

It is recommended that you make these changes after you have gone through your critical path task improvements. The reason is that by this point you will have a much better understanding of the underlying and connecting resources, which govern your cycle's

execution by having reviewed the critical path jobs. Just as working the critical path jobs first is a good strategy, working the "pretty critical" work and files next continues the same approach. However, you will not know what those "pretty critical" items are until you have spent some time on the critical path.

MAKE IT HAPPEN AND TRACK THE RESULTS!

OK, you are almost done. Now it is time to implement. It may be you have got a short list of changes, or perhaps you have pages of changes. No matter which it is, a project team with cross discipline participation never hurts and, for the sake of preserving the changes into the future, notification of a change so it is not undone by some future "innovation" is the only way to go. You will probably always need help from your Storage Management and Production Control groups. Unless you have found a coding change, which needs attention from your Application Support group, they may not need to be included. However, I would recommend including the Application group just for good measure. Forewarned is fore armed!

Develop a list of the changes you require and the relative importance (overall savings) of each change you are proposing. Be sure to include information on dependencies between changes. Listing dependencies is particularly important when applying changes that effect many jobs — for instance, moving a file from tape to DASD or into memory. You may not get all the changes you are recommending. Having an understanding of dependencies will help in ensuring the maximum benefit possible from the changes you can accomplish.

Finally, after the changes are in, be sure to track the batch to verify the results you expected. This is where a line chart of completion time overruns produced in an automated mode is a real time saver. It is also an excellent source for finding other cycles whose end times are trending in the "wrong direction." You may also find from your first few attempts that the estimated savings you used in modeling the changes do not hold water. That is OK; the next time you will know a better estimate to use!

CONCLUSION

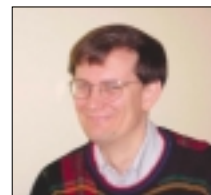
Batch tuning can be an overwhelming project from the start. However, by keeping

a disciplined approach to the tuning process, you can take on the analysis of a complicated system of thousands of jobs and turn it into a very manageable endeavor.

Just establish and stay with an approach that focuses on fully defining the objective. Determine up front if the objective requires a technical solution. If it does, be sure you understand what qualifies as "success." Identify a reliable base model of a troubled cycle and find the critical path(s) leading to the objective(s). Not all instances of a cycle are created equally, so be sure you are pulling data from the instances that reflect the problem.

Restrict your tuning to critical path jobs and keep the critical path updated. Do not bother tuning jobs outside of the critical path, as they typically do not have a direct bearing on your objective. As you determine the effect of improvements, update the model and determine if the change re-routes the critical path through other jobs. Always work the critical path jobs wherever they are. Coordinate the implementation of your changes and verify true results. Sometimes changes you chose are dependent on other changes and have a diminished effect if not co-implemented. Likewise, some changes may be so revolutionary that for the sake of their permanence you will need wider buy-in from your organization.

A disciplined and persistent hunter who has a plan of attack and sticks to it can find a lot of opportunity! 🎯



Wes Scott has been involved in the IT industry for more than 20 years. While his background has covered all aspects of data center configuration, management and tuning, recently, he has been involved in offering batch tuning services and products to Fortune 500 companies. His company's premier batch management product, VisiTrac, was used for examples in this article. He can be reached via email at wes.scott@lexonix.com or at (440) 826-0242.

©2002 Technical Enterprises, Inc. Reprinted with permission from **Technical Support** magazine. For subscription information, email mbrship@naspa.com or call 414-768-8000, Ext. 116.