Speed Up High-Throughput Business Transactions with Parallel Processing — No Programming Required!

Susanne Janssen and Werner Schwarz

Susanne Janssen, Performance & Benchmark Group, SAP AG

Werner Schwarz, IBS Consumer Industries, SAP Retail Solutions

(complete bios appear on page 94)

Parallel processing helps to improve the throughput and processing time of business transactions that must process lots of data in a tight time frame. As most of you know, parallel processing refers to a program's ability to simultaneously process individual packages of a worklist. But you, or the users you support, might not be aware that many high-throughput SAP programs actually offer a "checkbox" for enabling parallel processing. Members of your company's SAP IT team, performance specialists, consultants, and even power users can enable this option, instructing the system to process the discrete workloads that comprise that application in a parallel, rather than sequential, fashion. In so doing, they achieve greater throughput in a drastically reduced amount of time.

Of course, not all high-throughput applications offer this option, and even for those that do, parallel processing isn't always the remedy for sub-optimal performance. So, in this article we offer advice on:

- How to determine whether parallel processing can solve a high-throughput application's poor performance.
- How a high-throughput application with a built-in parallel-processing capability splits up the workload to achieve greater throughput, and what options are available to you to maximize its performance.
- When parallel processing appears to be a viable remedy to slow throughput, but the program doesn't inherently support this capability, how concurrent batch jobs can be used to boost throughput.
- Lastly, how the two most common pitfalls of parallel processing can be recognized and either avoided or corrected.

Are You Sure Parallel Processing Is the Answer?

When the jobs that comprise a program take too long to run, don't be too quick to invoke parallel processing or batch jobs. Do a sanity check first. Make sure that the overall construct and customizing of the business process are in order; if they are, check the code for gross inefficiencies. You'd be surprised what performance surprises lurk there!

Our team recently had the opportunity to assist a manufacturer of domestic appliances with an ailing Make-to-Order Backflush. In layperson's terms, the Make-to-Order Backflush is a batch job that looks at the bill-of-material line items inherent to a product and then initiates the orders that get those items transferred from stock to a manufacturing facility. In this case, the Make-to-Order Backflush was looking at the things the people in the assembly lines had to have on hand (such as stickers, screws, etc.) to produce a particular appliance. It was initiating the transfer of those materials from stock to the manufacturing facility via something called "Materials Movements." In the SAP system, every Materials Movement leads to follow-on Material documents and Financial documents. Upon closer inspection, we found that this customer's system had been customized in a very non-efficient way. Customizing settings were triggering Material documents and Financial documents (*lots* of them!) for inconsequential materials. There were even documents for drops of oil! The processing of these documents was consuming an excessive amount of system resources.

Not only was the *customizing* inefficient, the *construct* of this business process was just plain wrong. (This customer's best bet was to use the Bulk Material process instead of the Make-to-Order Backflush.) Until these kinks were worked out, turning to parallel processing for this job's work packages would be premature.

> ✓ ***The lesson we hope to impart with this example is this…***
>
> *When trying to ascertain the root of a high-throughput application's performance problem, look at the big picture. Has the right SAP solution been invoked? If it has, is the customizing in order? If you can answer both of these questions with a resounding "yes," then it makes sense to invest time, money, and energy digging into the code to unearth performance problems there.*

If the Construct and Customizing of the Ailing Business Process Aren't the Culprits, Check the Code

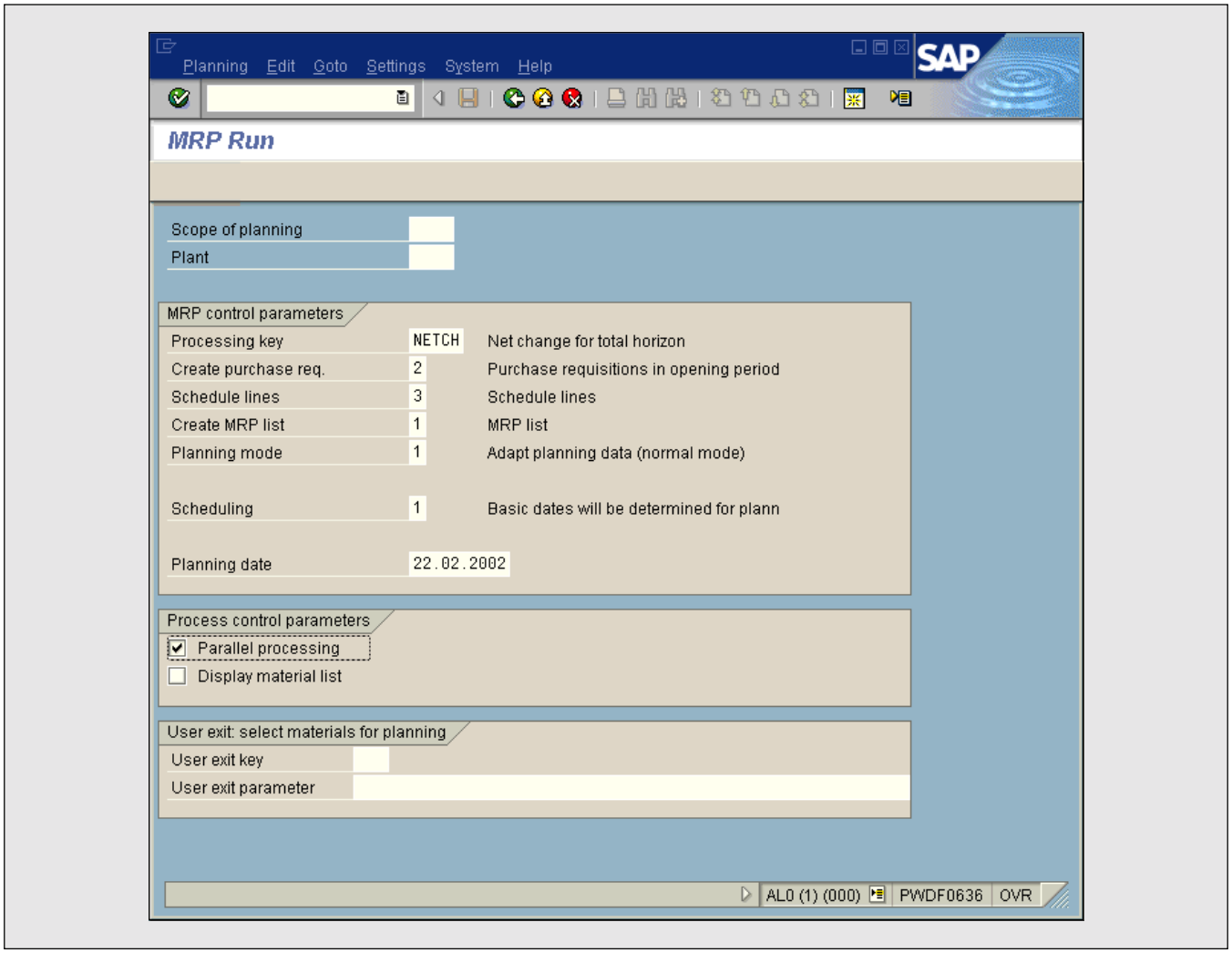
Had we not found any problems with the construct and customizing of the business processes associated with this customer's ailing Make-to-Order Backflush, we would have examined the code itself. An

Figure 1 Top Five Programming Performance Traps (at DB Level)

- ☑ There are missing indexes on the database.
- ☑ Customer exits are filled with inefficient SQL statements.
- ☑ Identical SELECT statements exist within one transaction, causing several accesses to the same data and thus more loads on the database than necessary.
- ☑ The program fails to leverage SAP's buffer mechanism. The result is that data needs to be retrieved from the database instead of from the application layer, creating a bigger drain on system resources.
- ☑ Database calls have incomplete WHERE clauses, which cause the number of data records loaded from the database to be much higher than actually required.

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

Figure 2 The Parallel-Processing Option for the Material Requirements Planning Transaction



application's code offers a seemingly endless number of possibilities for slow performance! You'll want to focus initially on the most likely culprits. In **Figure 1**, we offer a quick summary of the performance traps we suggest you look for first (or what you should instruct a developer to look for on your behalf) in order to rule out bad coding practices as the source of poor throughput. Much more complete and comprehensive guidelines can be found at **http://service.sap.com/performance**.

Once you've ruled out the three Cs — **Construct**, **Customizing**, and **Code** — as the sources of troubled throughput for your high-volume application, it's time to turn to parallel processing. Or is it?

The option to invoke parallel processing does not exist for every high-throughput program. It is only available in applications in which a software developer has built parallel-processing provisions into the code. We wish we could present a definitive listing of every SAP program that offers the parallel-processing option. It's just not possible. What we can tell you is that most applications that were designed from the start to handle large data volumes *do* offer the option. But not all.

In applications that do offer the option, the user is usually presented with a checkbox to invoke parallel processing, as is the case with the Material Requirements Planning (MRP) run shown in **Figure 2**. In

this figure, you can see that we checked the box to enable the option.

What about applications that don't offer this option? And there are quite a few.[1] For these applications, you can employ several batch processes to achieve parallel processing of the workload.

So when you have determined that parallel processing is the right plan of attack for your ailing high-throughput application, you'll be headed down one of two paths:

- The ailing application has a built-in option for parallel processing, of which you can take advantage.
- The option for parallel processing was never built into the application, so you'll have to enact parallel batch processes to achieve better runtimes.

We'll show you how to optimize your program's runtime for both situations in the sections that follow.

Optimizing Runtimes for Programs That Do Offer Users a Parallel-Processing Option

In **Figure 3,** you see the Store Replenishment application from SAP Retail, which manages a typical high-volume transaction and thus *has* been designed by SAP with a parallel-processing option. In this screen, you can see that to invoke the option, users simply need to click on the "Use parallel processing" checkbox, which is located in this application's "Technical options" section.

Note that the parallel-processing option is usually located in the "Technical options" section of the transaction, but not always; not all transactions provide such a section. The location of this checkbox isn't standard across applications — it's situated at the discretion of the application's developer. Sometimes it is located on the initial screen, sometimes on a secondary screen. If the location of this checkbox isn't readily apparent, check the documentation or SAP Notes.

An application's developer also exercises discretion over how the workload is to be split up once the option is invoked. That is to say, the developer picks the split criterion (it varies from application to application) by which the total workload is divvied up into smaller, discrete, non-overlapping workloads. In addition, the developer may decide to allow the application users to influence the size of workloads and where processing takes place by providing options like the ones you see in Figure 3: "Number of recipients per process," "Logon/server group," and "Max. number of processes." Once the workload has been divided into smaller work packages, the program assigns those packages, via remote function calls (RFCs)[2], to multiple R/3 instances for processing.

While users don't need to understand the intricacies of RFCs to benefit from invocation of the parallel-processing option, they do need to understand the relationship between RFCs and *server groups*, because it is via server groups that RFCs are assigned to instances for processing. The server group the user selects for parallel processing ("Logon/server group" in Figure 3) has an impact on the application's throughput. In addition, with the proper authorization, the user may also be able to define the server group itself and thus further influence performance.

In the sections that follow, we'll provide you with the information and insights you need to get the most out of the parallel-processing options an application's developer has provided.

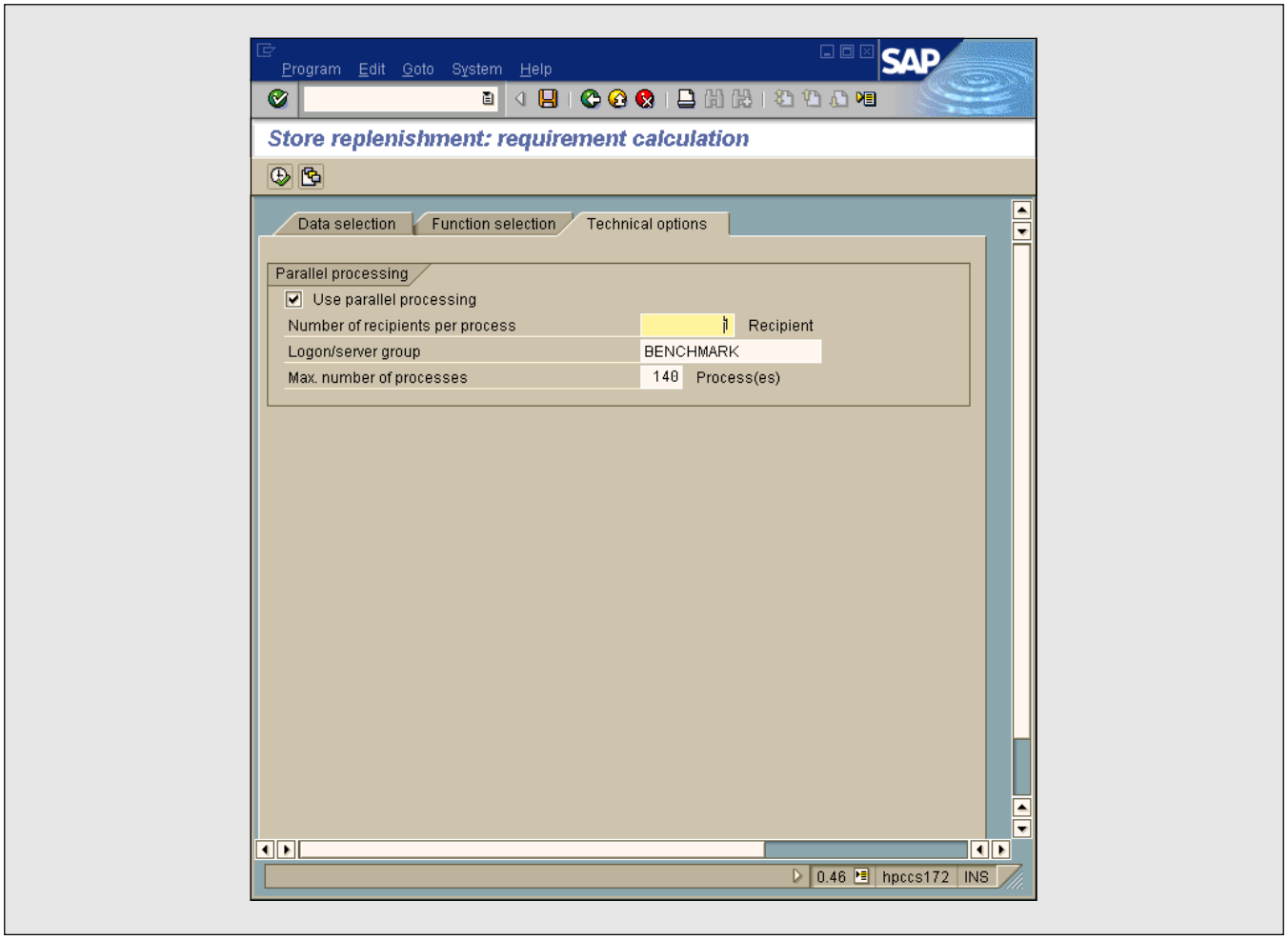
---

[1] How could this omission be made? Our guess is that the developers did not anticipate the applications would be used to process large volumes of data.

[2] In the SAP system, the ability to call remote functions is provided by the Remote Function Call (RFC) interface protocol. RFCs enable developers to write programs that call and execute predefined functions between any two SAP systems or between an SAP system and a non-SAP system, or even between functions within the same SAP system. There are several kinds of RFCs — asynchronous (aRFC), transactional (tRFC), queued (qRFC), and serial (sRFC). In the context of parallel processing, we are concerned with aRFCs.

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

Figure 3 Invoking the Parallel-Processing Option in Store Replenishment



What You Need to Know About Server Groups

A server group is a logical group of R/3 instances. In any given R/3 system, those with the proper authorizations within the Computing Center Management System (CCMS) can define as many server groups as needed, then assign instances to those server groups. A server group can consist of one or more instances, and the same instance can be assigned to several server groups.

In an application with the parallel-processing option enabled, an aRFC (asynchronous RFC) is started to facilitate the processing of a work package. The aRFC typically specifies a server group (which the user may select if the option is available, as in Figure 3), and a dialog work process[3] is used on an instance of that server group for the processing of that job.

The great advantage of using server groups is that the *system* can assume responsibility for picking which instance within a server group should be invoked. The administrator, performance specialist, or power user who is invoking the parallel-processing option does not need to worry about prevailing system loads and which instance should ideally get picked to process a work item.

[3] A dialog work process is a type of work process dedicated to processing programs with user interaction (i.e., online transactions requiring user input).

If you have the proper authority, however, you can influence an application's performance by adding or deleting instances (i.e., processing power) for a particular business process simply by changing the server group definition. You can also set certain value limits that establish whether the system can push more work onto an instance, or whether the instance should be regarded as fully utilized. We show you how to do these things next. At no time do you ever need to alter the selection variant or the program's code.

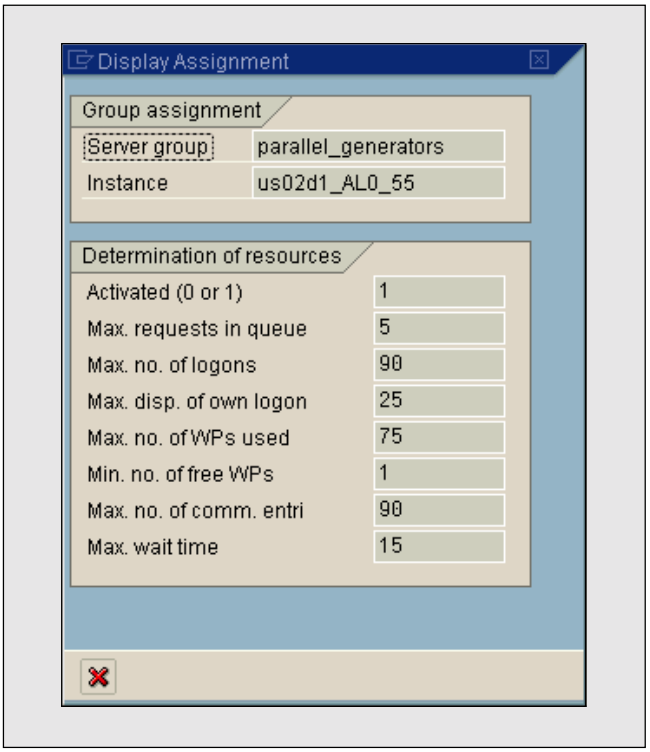
> ✓ ***Tip***
>
> *With proper authorization, you can control which instances get mapped to which server groups. This means you can ensure that specific instances are used for specific processes. For example, you may want to make sure that helpdesk staff at a call center have ample processing resources dedicated to the applications they need in order to ensure fast response times. By keeping all the instances that should be reserved for the helpdesk staff out of the server groups that are used for parallel processing, you can ensure they have sufficient resources.*

Defining a Server Group

Assuming you have the proper authorizations, you can use transaction RZ12 to define a server group, assign instances to the group, and as of Release 4.5, temporarily define the value limits for the load situation of the instance, as shown in **Figure 4**. Here, you can see that we have assigned instance "us02d1_AL0_55" to an example server group called "parallel_ generators." You can choose any name for a server group you like; we recommend using a name that refers to the process or group of functions you want the server group to handle.

Just beneath the "Group assignment" section in Figure 4, you see a section labeled "Determination of resources." The parameters set here influence the system's assignment of workloads to this instance.

Figure 4 Display Assignment for Server Group



Note that the values for many of these parameters are percentages, not absolute values. Exceptions are the parameters for general activation/deactivation of the value limits, the parameter used to define the minimum number of free dialog work processes, and the parameter for the waiting time (as of Release 4.6). Detailed information on the individual parameters can be found in SAP's online help and the SAP Library.

You can promote efficient load distribution by setting the parameters that control the number of instances or free work processes (WPs):[4]

✓ **Min. no. of free WPs:** This parameter, which specifies the minimum number of free work

[4] Unless you are very familiar with the technical details of the RFC interface system administration in R/3, we do not recommend changing any of the other parameter settings.

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

processes, must always have a value of at least 1. However, experience has shown that it is better to leave more than just one process free (we recommend three, at least). For many reasons (e.g., filling the number-range buffers, additional RFCs within the program, monitoring, and so on), the system often needs free dialog work processes for short periods. For example, if no free work process is available to fill the number-range buffer, processing time increases because the system must wait until a dialog work process is free. You may have to set up additional dialog work processes in operation mode for runtime so that the number of processes that you want to use for parallel processing and the free work processes the system requires are both available.

✓ **Max. no. of WPs used:** You also may have to increase the value of this parameter if the ratio of work processes that you want to use to the total number of dialog work processes is higher than the percentage assigned to this parameter.

The fact that you assign an instance to a server group *and* define value limits for the load distribution on that instance in the same transaction (RZ12) often leads to misunderstandings. The most common misconception is that the value limits entered for the instance in this transaction *only* apply when the instance is included in the particular server group it is assigned to in this transaction, and that other parameters can be defined for the instance when it is included in other server groups. However, there is only *one* set of valid parameters for each instance. The parameters that take precedence are the ones that were set most recently.

This means that the parameter changes you make when you assign an instance to another server group will overwrite the parameters you previously entered for the instance. For example, suppose you have two transactions that will use parallel processing and that these transactions will run at different times, one during the day, one at night. In an effort to make use of all available processing power for both transactions, you assign instance A first to server group 1, which is used at night; then you assign this same instance to server group 2, which is used during the day. If you modify the parameters of instance A when you associate it to server group 2, those changed settings will also apply when instance A is used as a part of server group 1.

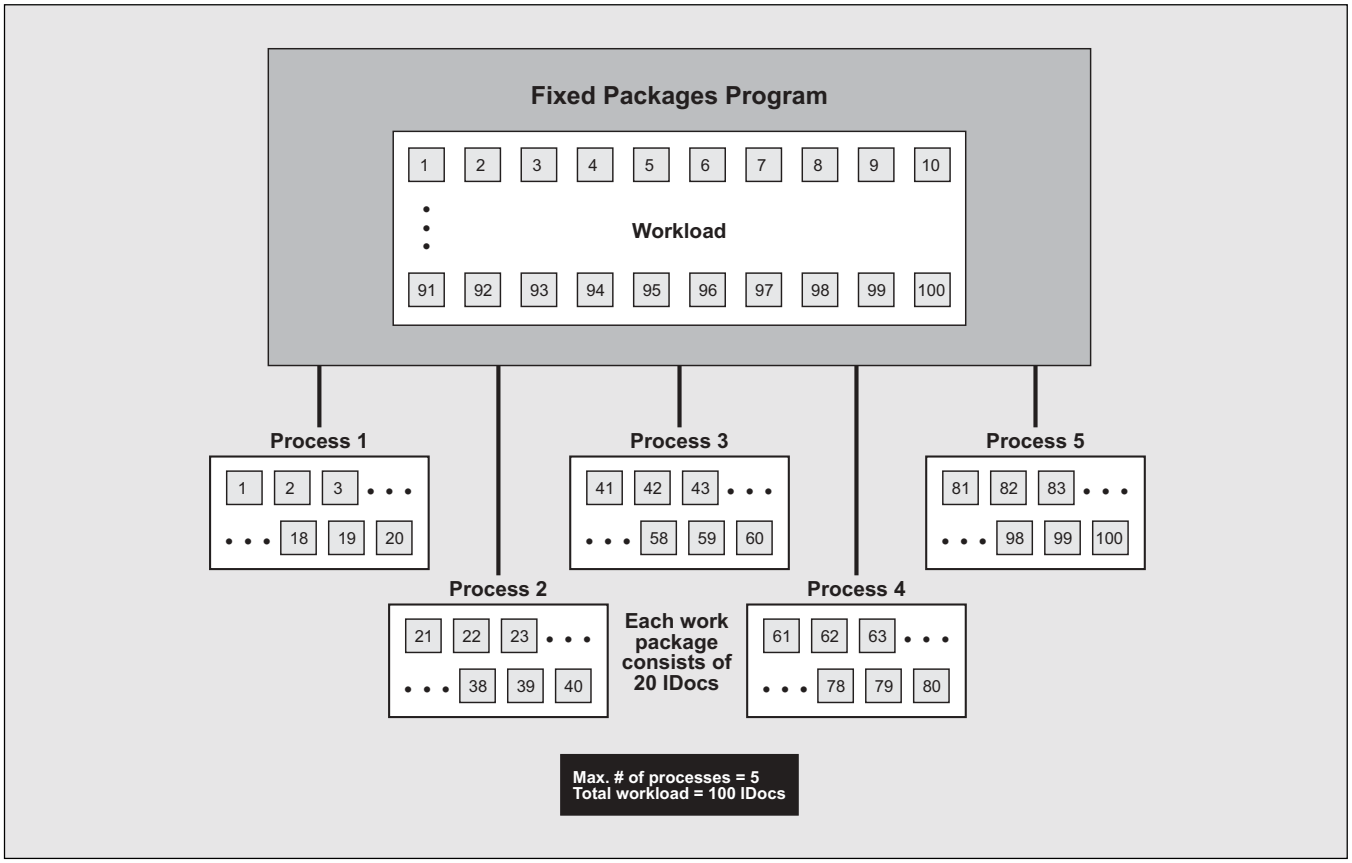
This restriction (only one parameter set for each instance) generally does not cause problems, because the settings of the parameters are usually oriented to the hardware configuration (number of CPUs, amount of memory) of the application server. As this does not change, there is little need to use different parameter sets, except when the way you set things up most recently (in the example, for a daytime run) might not be the way you want them set up for another run (the nighttime run).

> ✓ ***Tip***
>
> *When arriving at suitable limits for the "Determination of resources" parameters, you should always take into account the operation modes of the system: The server groups and the value limits for the instances are usually defined during the day. Programs for parallel processing of mass data, however, often run at night! The operation mode for nighttime processing is often configured with more batch work processes and fewer dialog work processes than for daytime processing. If you base your settings for an instance's resource parameters on the current (daytime) operation mode, rather than the operation mode for the program runtime (nighttime), the overnight program may use fewer resources than you had planned. You should therefore adjust the night mode accordingly.*

Splitting Up a Larger Workload into Smaller Ones

So much for optimizing the environment in which the parallel workloads will be processed. Let's step back now and examine how developers split up a large

Figure 5 *Fixed Packages*



workload into smaller ones (this influences how you might use choices offered with parallel processing). There are two options:

- The program splits up the workload into fixed packages.
- The program assigns small packages dynamically.

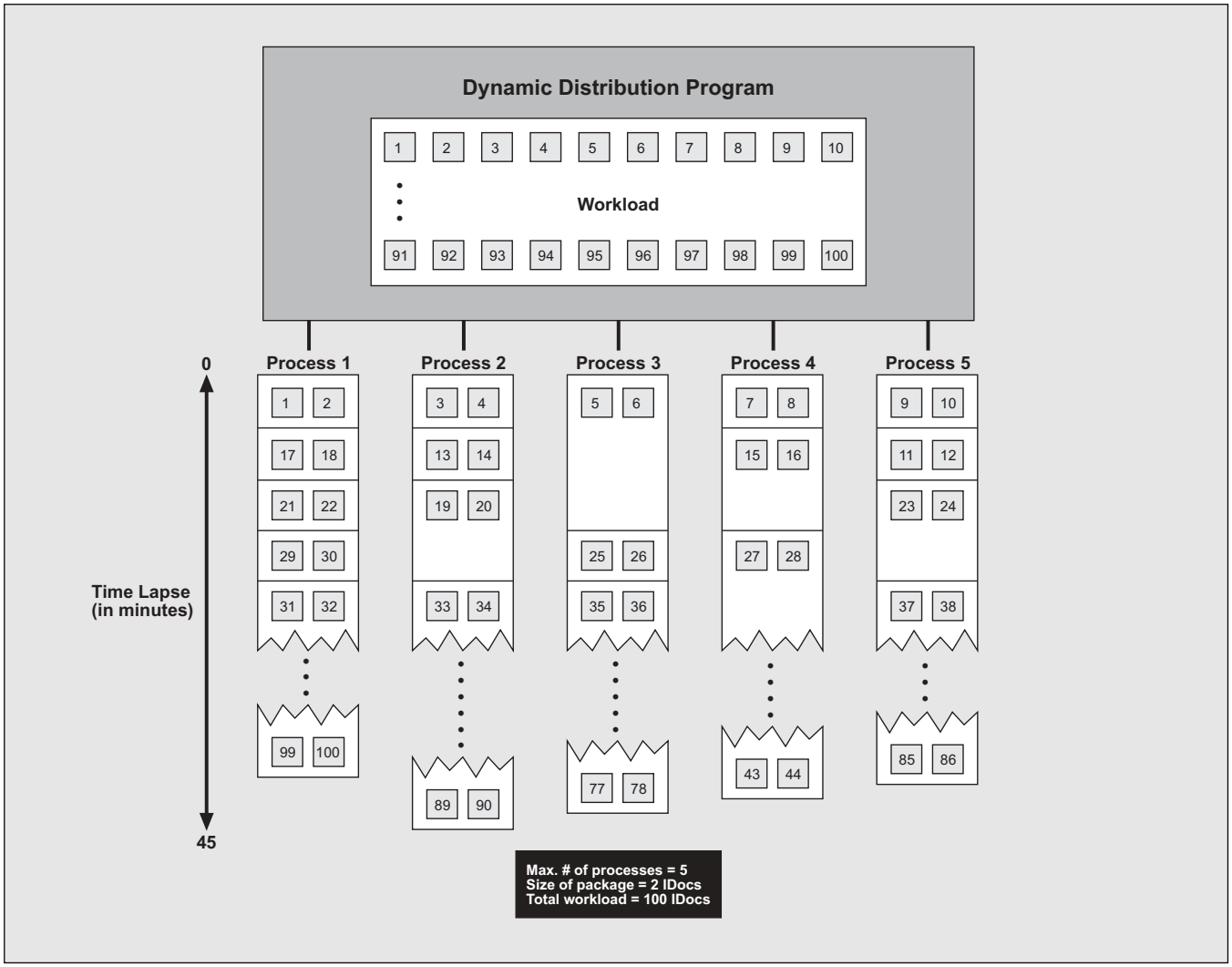
With the first method, where the program splits the workload into fixed packages (the less common and less desirable of the two), the developer has a program analyze the current workload and divide it into packages according to some suitable split criterion defined by the developer. In a Retail application for a nationwide chain of stores, for example, the workload might be split up according to the individual stores in the chain. In this case, the Intermediate Documents (IDocs) used to upload sales data into the central system are split up according to the stores from which they come. This split criterion will ensure that a store's data is not processed in parallel (as this could cause locking conflicts).

If there are 100 IDocs, and five processes available for parallel processing (the number of processes available can be determined either by the developer, the application user via a selection screen, or the number of available resources in the system), the IDocs get evenly split among five aRFCs. IDocs 1-20 would then be assigned to one aRFC, IDocs 21-40 to another aRFC, and so on. In a Financials application, the split criteria might be pay-to intervals.

The advantage of this method is that a large load can, in *theory*, be distributed evenly across instances, as shown in **Figure 5**, and it can be implemented quite easily.

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

Figure 6 *Dynamic Distribution*

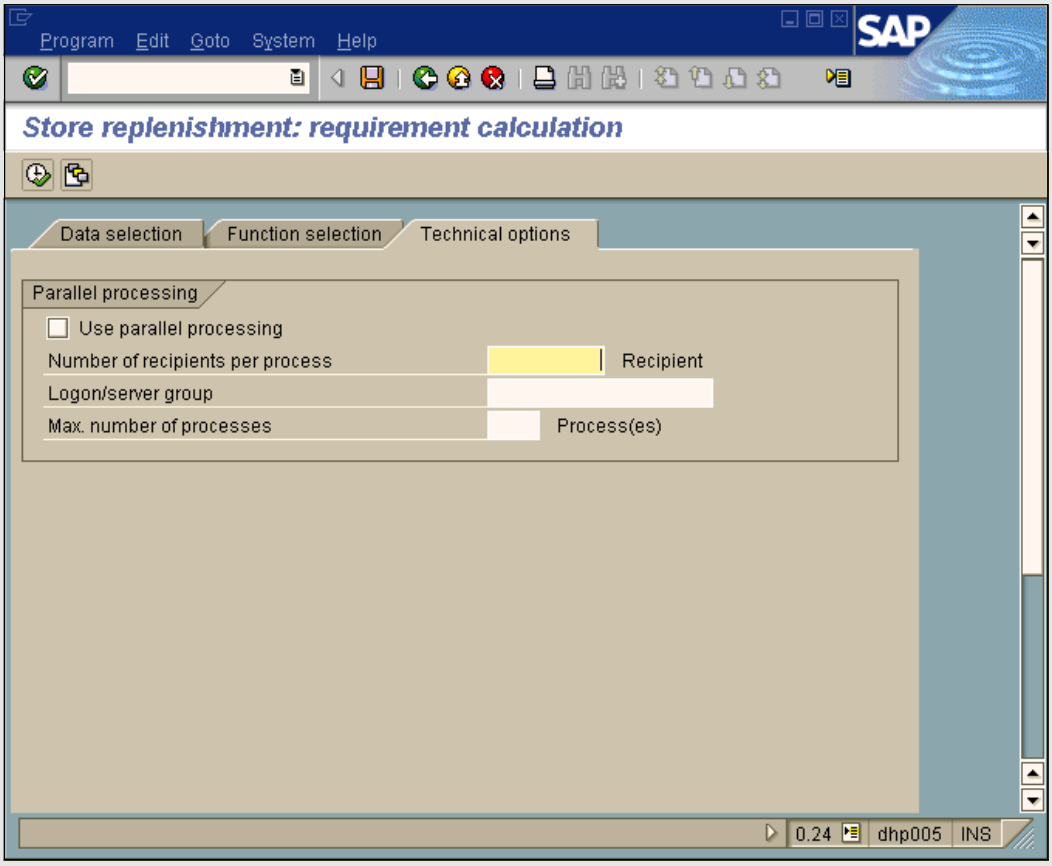


However, as we all know, theory and reality often don't mirror one another. How can the developer be sure that the workloads associated with each package (IDocs 1-20, 21-40, ... 81-100) will be roughly the same size? If the packages are all the same size, everything's fine. But, consider that we're talking about different stores in different parts of the country. Some stores may have lots more sales data than others, causing different size workloads to be distributed across the instances. If IDocs 1-20 are all very large, while the rest of the IDocs are all very small, IDocs 1-20 are all going to be processed by the same job allocation, even if the jobs set aside for the rest of the IDocs are idle. This larger load will take more time, and when it does, follow-on processes are likely to be affected (i.e., they will suffer a performance hit). So, while there are certainly runtime improvements in that not all 100 IDocs are being processed by the same job, this scenario is still subject to gross inefficiencies (and for this reason is not widely adopted).

Now let's examine the more commonly used method, where the program determines and assigns data dynamically. This method is a bit smarter and evaluates things as it goes along.

Take a look at **Figure 6**, and then look again at Figure 5. Notice in Figure 6 that very small packages

Figure 7 Options for Parallel Processing



(2 IDocs each) are being parceled out to the jobs, one at a time. A customer's IDocs are not pre-assigned to specific jobs. A program starts one aRFC (or a package of a few) for each IDoc object until all objects have been distributed or no further aRFC can be started, because either the system resources are exhausted or the maximum number of jobs set by the developer or the application user is reached. The communications management of the aRFCs provides an easy means to recognize the end of a job; as soon as processing resources (server group work processes) are available again, the distribution continues. The individual job packages are small, so excessive memory consumption is avoided.

This method of workload distribution has a good chance to achieve an (almost) equal load for all processes. The time to reach full completion of work is minimized, and no system resources ever remain unused. Even if the individual work packages have drastically different processing requirements (i.e., some will take much longer than others to process), dynamic distribution (to the next available process) ensures that the next job will be assigned to the processor with the smallest load so far.

User Options for Influencing Workload Distribution

Take another look at the selection screen for enabling parallel processing that we showed you earlier (for easy reference, it is shown again here, as **Figure 7**). If you activate the checkbox for parallel processing

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

here, then you must enter values in the other fields as well — i.e., you will choose the server group to be used, how many parallel jobs are to be used at most, and how many work items should be processed in one job (= size of package).

The application that you use may offer parallel processing as the only option, it may offer only one additional option (maximum number of processes), or it may offer all of the options you see in Figure 7.

Does this mean that you can tell by the options that are offered whether your application was designed with fixed or dynamic assignments? Unfortunately, there is no easy way to tell. Analyzing the code (or getting a developer to do it for you) is the only sure way to know. Even the existence of a field for you to enter package size doesn't necessarily mean that the load is assigned a certain way.

If the developer has provided an option that allows you to specify the number of processes the packages run on, the value you enter here represents an upper limit, as the program can use only the number of jobs that the system allows.

If the program provides dynamic job distribution, the user usually finds an option to set the maximum size of a single package (in Figure 7, it's "Number of recipients per process"). If there is no such field on the selection screen for parallel processing, the user can assume that the program probably does not use dynamic load distribution.[5] To set package size for dynamic distribution, the user must enter a value in the field for the maximum package size.

Assuming the user can adjust package size, what is the optimal package size? Unfortunately, there is no answer that applies to all situations. However, we can say that if you decide to use very large job packages by entering a large value in the "Number of recipients per process" field, you are likely to have the same situation as when a load is distributed in fixed packages. If the packages are large and the workload of each object is roughly the same, a good load distribution can be expected, but (as we discussed earlier) if this assumption is wrong, and a larger-than-expected load takes more time to process, follow-on processes will be delayed. In addition, the program needs more memory for large jobs, so limited memory may become a bottleneck. Also, consider that with the longer runtime required by a large load, time-outs might be a problem.

In cases where you run into memory problems, our recommendation is to first decrease the package size, which should reduce the memory used per job. If this doesn't help, reduce the number of jobs on the application server, so that the total amount of required memory is reduced.

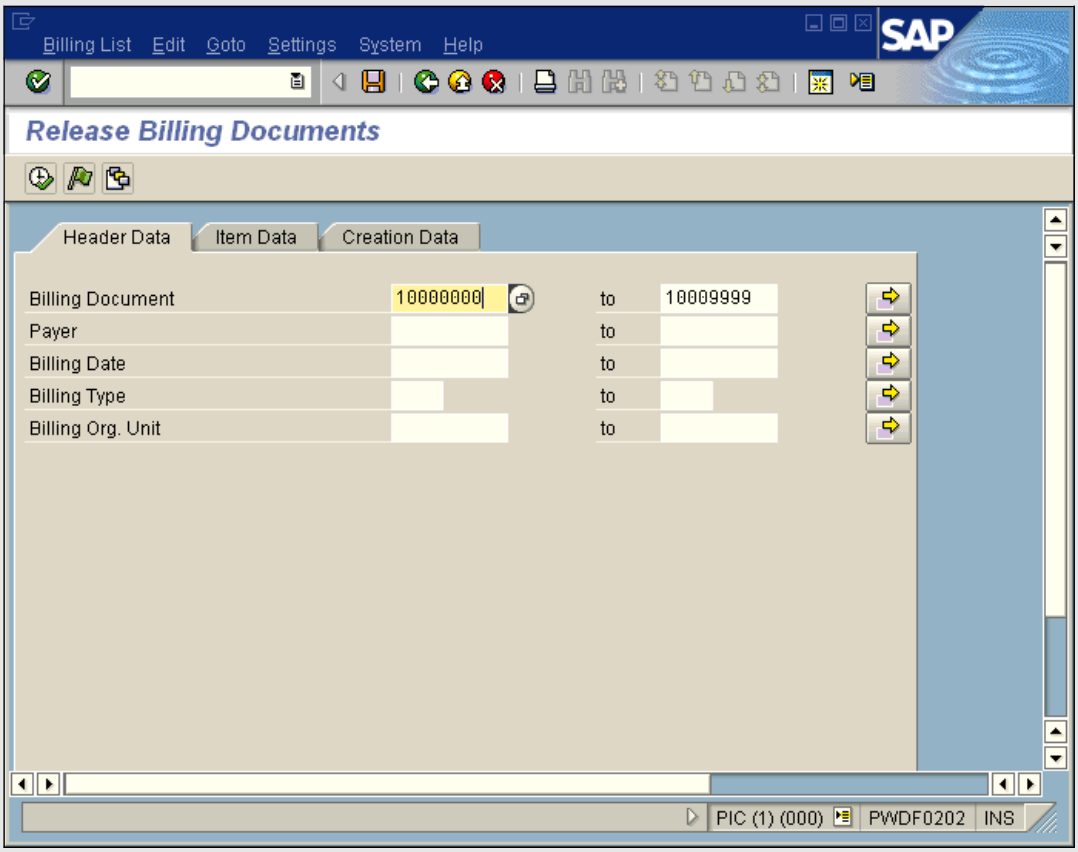
Invoking Batch Jobs for Programs Without a Parallel-Processing Option

If you've ruled out the three Cs — Construct, Customizing, and Code — as the culprits for poor throughput and determined that parallel processing is indeed the right antidote for your ailing application, what do you do when a parallel-processing option does not exist for that program? You divvy up the workload into smaller work packages yourself, then get multiple concurrent batch jobs going to process those work packages in parallel.

Before you attempt to do this, you must have a solid understanding of the type of load you're dealing with. Suppose, for example, that you are a retailer. This time, we'll assume you've got 1,000 stores, all of which need to send Point-of-Sales (POS) data back to the central system for processing. To be able to handle this load, you need to plan for several batch jobs and pick an appropriate criterion for splitting the workload into smaller work packages. Then you can define the batch jobs and assign selection variants in the selection screen.

[5] But as we said earlier, if there is no such field, you cannot be sure that distribution with fixed packages is used.

Figure 8 *Billing Transaction*



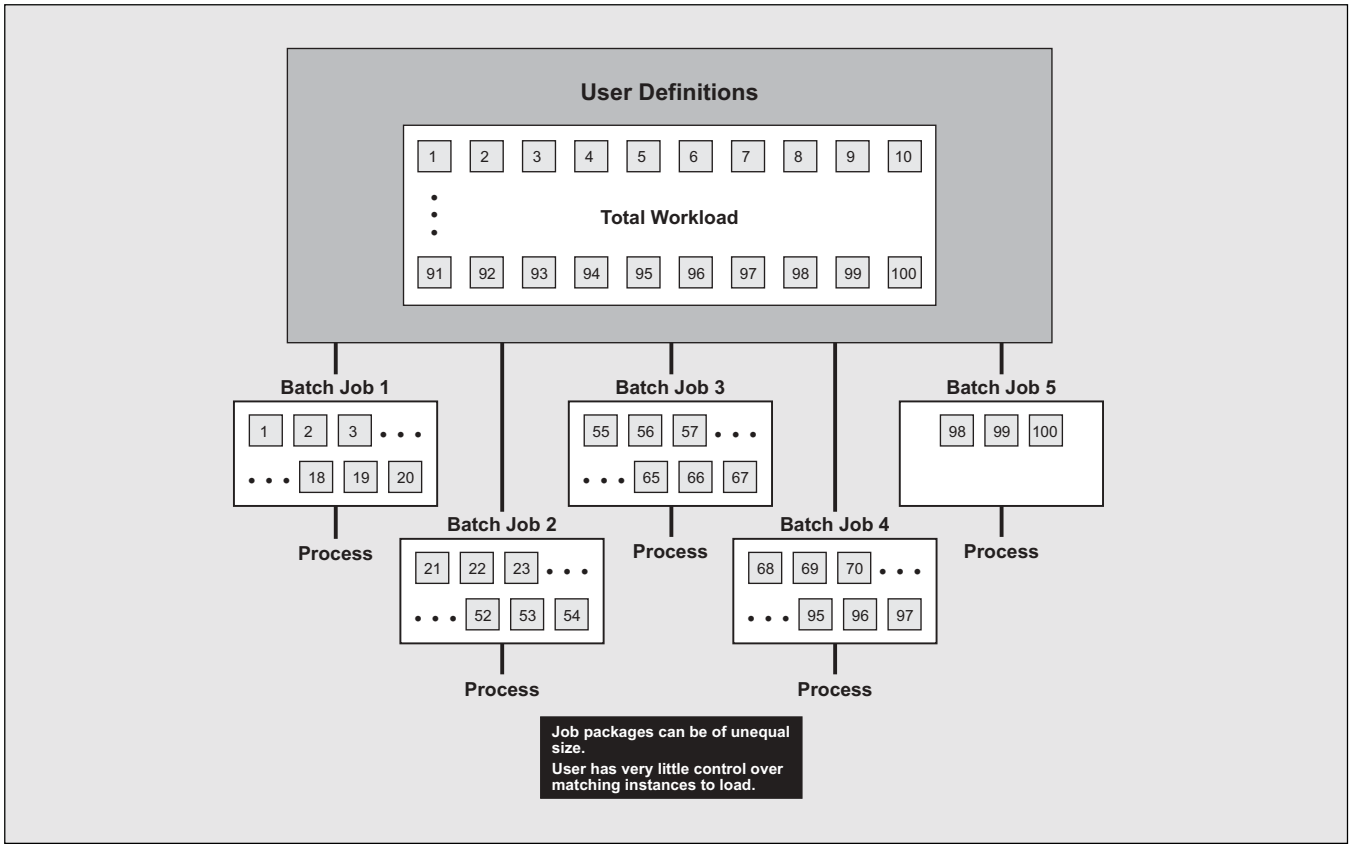
The selection variant then assigns to each of these batch jobs the IDoc data that needs to be processed. If the package loads are fairly predictable, this method can work well. Because you define the selection variant only once (and maybe only make minor changes when store information changes), the ongoing administration effort is minimal. Also note that these batch jobs will only need to be planned once.

Using the CRM Billing transaction Transferring Billing Documents to Accounting, you can transfer billing documents with a transfer block, or you can manually transfer incorrectly transferred documents to the active accounting components. In **Figure 8**, you can see that in a mySAP CRM billing transaction, we have elected to divide things up according to the number of billing documents.

> ✓ ***Tip***
>
> *We recommend you use relatively stable data as selection criteria, such as stores or bill-tos. To define changing data as a selection criterion is not a good idea. For example, we do not recommend using numbers of IDocs, deliveries, or orders because these constantly change.*

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

Figure 9 User Defines Batch Jobs of Almost Equal Load



Imagine what could happen if the stores in our example are of different sizes, however. The loads for the packages could differ quite dramatically, with the result that there might be a few long-running jobs still executing after all other jobs finished their work. To avoid this scenario, you would have to judge the runtime caused by each IDoc in order to define packages of almost equal load (see **Figure 9**). But the load from each store can differ from day to day, so that you would have to perform this analysis every day. Furthermore, the amount of data sent in from each store may change significantly every day. Still, it beats having no parallel processing at all!

You have only two ways to assign batch jobs to instances:

- You can assign one job to one instance when configuring your batch job as a background job. In this case, the job can only run on the assigned instance. If there is no free batch work process at the time you planned the processing, the job will wait until another batch work process on the same instance is free again. You cannot make adjustments based on the actual load of the instance. If the instance is under heavy load, the planned job is processed on the same instance in addition to the existing load, even though other instances may have more resources available to process that job.

- Alternatively, you can let the system assign the batch job to a particular instance. This still doesn't guarantee that all the instances will bear the same load or that the job will run on an instance that has sufficient capacity. Particularly annoying is the fact that you cannot partition off particular instances and allocate those resources to specific tasks. The only methods available to you are either to use the job class to give basic

directions for the jobs, or to exclude instances from any batch work processes. The definition of a job's class is performed during the configuration of the background job. The exclusion of batch processes from an instance would have to be done by the system administrator.

Note that this manual approach to parallel processing can consume quite a lot of memory because the data packages tend to be quite large.

Two Common Parallel-Processing Pitfalls

If you have changed a program to run in parallel (by invoking a built-in parallel-processing option or by farming out the workload to concurrent batch jobs), yet you notice a longer-than-expected runtime, you will need to play detective to discover the reason. The two most common reasons are locking conflicts and resource constraints. In this section, we review these performance pitfalls.

Performance Pitfall #1: Locking Conflicts

What do we mean by a "locking conflict"? Whenever there is a lock for a resource held by one process, and another process is delayed or hindered in its work by this lock, there is a locking conflict.

For example, whenever a process modifies a record on a database, the database system locks this record until the modification is either committed or rejected. If another process tries to modify the record while it is locked, this new modification is delayed by the database system until the lock on the record is released through the commitment or rejection of the modification by the first process.

Locking conflicts can occur in other areas as well. In SAP R/3, a lock may be set for a specific system resource, e.g., enqueues (also known as SAP locks). The behavior of a program is different under different kinds of locking. Let's look at the two major types, enqueues and DB locks:

✓ **Enqueues:** When there is a locking conflict involving an enqueue, the processing of the current work item is usually canceled. In this case, the runtime of your program does not increase; instead, you'll see that not all objects have been processed. The process must be set up again for the missed objects, which takes more time than you can afford in a tight schedule. This type of locking conflict can occur in common circumstances, such as when you want to update the material stock for the same materials in the same plant from several parallel processes. Use transaction ST05, the Trace Request, to observe where locks are set. Once you find the enqueue, see if you can either avoid the conflict by changing the split criterion or get the developer to make adjustments to the program design to avoid invoking this lock.

✓ **DB locks:** Conflicts that take place because of database locks lead to higher runtimes, because the second process is waiting until the lock of the current process has been released. DB locking conflicts can occur with any database table that is updated. You can detect locking conflicts on the database at runtime with the transaction monitor DB01. A good development practice that helps avoid or clean up these conflicts is to analyze the pattern of data changes in the program and any others that run at the same time. In addition, the developer should check to see whether the criteria for splitting workloads is appropriate when the same program runs in parallel in several jobs. Finally, we often find in SAP R/3 that DB locking conflicts occur with the assignment of unique numbers (this is discussed in the sidebar on the next page).

Performance Pitfall #2: Believing That N Parallel Processes Lead to an N-Fold Increase in Throughput

During parallel processing of a large workload, one

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

Avoiding Number Assignment Locking Situations

Unique numbers are often assigned to documents (such as new delivery or order documents), as well as to IDocs. In general, the system determines the next unused number from within a given number range or interval, then makes the assignment. Sometimes, however, a process stops after it has requested the number assignment, but before the document could be saved. In this case, the number has been specified but no document bears its assignment. This may not present a problem for most of you, nor for most kinds of documents. If, however, you live in a country like Italy, where the taxation authorities prohibit gaps in the number ranges of financial documents, you need to take special precautions.

To meet the most stringent requirements for unique numbers, the default settings will often mark the last number of an interval as "assigned" only when COMMIT WORK is called. In the meantime, the entire interval is locked by the database and no other process can obtain a number from this range of numbers.

Since the locks can actually last quite some time, conflicts can occur during parallel processing. For each number range object, there is one entry in the NRIV table (an SAP database table that is required for assignment of number ranges) that contains the interval and the latest assigned number. When a new number is requested, the system locks this data record with a SELECT FOR UPDATE command. The lock can only be released by either a COMMIT WORK or a ROLLBACK WORK command. Now, when a parallel process requests a new number for one of its number range objects, the process must wait until the lock is released. Depending on the number of parallel processes and the amount of time the locks persist, the probability of locking conflicts on table NRIV increases.

If you are in a situation where unique numbering regulations are not too restrictive, you can change the default settings to avoid locking conflicts. With transaction SNRO (Number Range Object Maintenance — the ABAP Workbench transaction used to create and maintain number range objects), the developer (or the system administrator, or consultant) can define a buffer mechanism for each number range object. In such a case, a certain range of numbers is taken from the database table and stored in main memory. Whenever a number from this buffered number range object is requested, the next free number is taken from the buffer and handed to the application program. The system does not monitor whether there is a document created with this number, and it is always ready to immediately hand over the next number.

The SAP documentation provides information on other possible solutions to number locking problems.

might assume that throughput would increase by a factor equivalent to the number of jobs run in parallel: Parcel out the workload between two instances, process them in parallel, and you might expect to double throughput. Parcel it out to four instances and expect quadruple throughput. Parcel it out to *n* instances and expect an *n*-fold increase in throughput. Not so fast!

An *n*-fold increase in throughput can *only* be achieved if the work items can be processed by those *n* instances in a completely independent fashion. This means that there is no resource in the system that becomes a bottleneck when several processes have to use it.

Take, for example, a case where we assume the network connection between the application server and the database can transfer 100 MB per second. If our program has to read 10 MB per second, there is no problem. If we run this program in 5 parallel processes, there is still no problem. But you can imagine what happens if we run 20 jobs in parallel! The network becomes a bottleneck, the data transfer is delayed, and the runtime for the program increases.

If you have a situation where using more processes does not bring about a shorter runtime during your performance tests, you need to find out whether a system resource is the bottleneck that prevents you from increasing the throughput (assuming, of course, that you have already validated the three Cs — Construct, Customizing, and Code). If you are not able to remove this resource restriction (e.g., by obtaining better hardware, optimizing the program, modifying the process design, and so on), your parallel program will not be successful in improving throughput.

If a suitable limited number of processes are started per application server (depending on the number of CPUs) and the database server has enough resources to execute this number of processes, then it's usually the I/O system of the database and/or the enqueue server that becomes the bottleneck. To know if either is the culprit requires a technical performance analysis conducted by experienced analysts. However, we can give you a few hints about some of the monitoring transactions used for such analyses:

✓ Problems with the database settings can be found with SAP's monitoring transactions for the database. For example, transaction DB01 can reveal exclusive "lockwait" situations, which can often explain why programs appear to be doing nothing. With transaction DB02, you can check for missing indexes and many other issues that arise with tables and indexes. Database transaction ST04 provides high-level statistics; among its useful features, you can see if there are problems with the database interface that services the program's requests for data.

✓ The detail screen of transaction STAT may be displaying statistics that reveal unusually long amounts of time spent on enqueue requests. You will also find STAT useful for other statistics. In STAT, after a pop-up, you can delimit the output (e.g., users, time period, program) to get a list of all transactions with your chosen characteristics (see **Figure 10**); the results are sorted by response time, DB time, CPU consumption, memory consumption, etc.

If I/O is determined to be the culprit, you will have to work with the database administrator to optimize the system layout and/or the management for storing the database tables on the file system. If it is the enqueue server, the answer might be a better CPU, binding a process to the CPU, and so forth. Alternatively, a developer can customize the application to reduce the number of enqueues by taking actions such as limiting the size of the Materials Management documents.

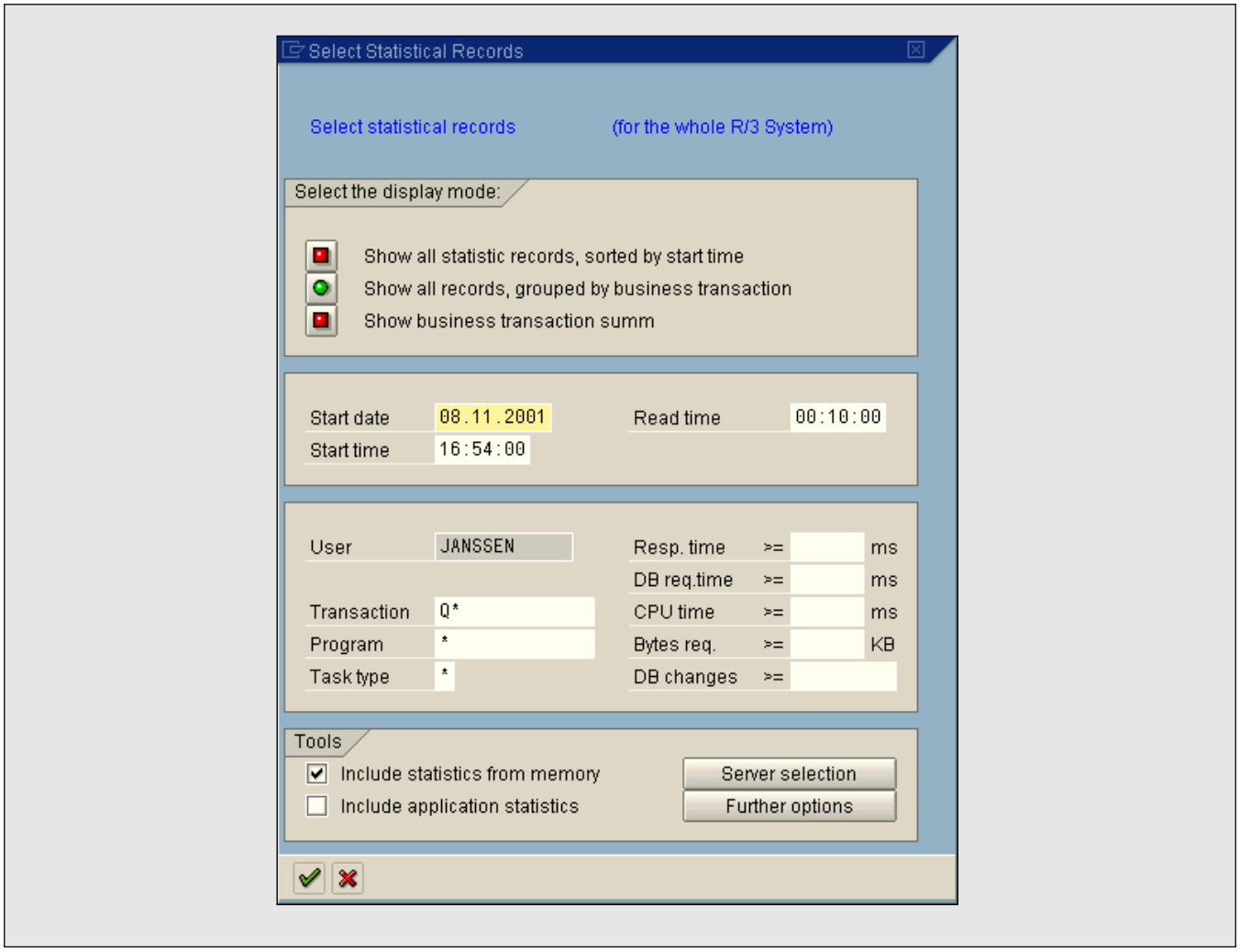
Conclusion

In the course of this article, we have made several recommendations for using parallel processing to solve the problem of a high-throughput application that is taking too long to complete. Here is a distillation of the lessons we hope you take with you:

✓ Before attempting parallel processing in either of its incarnations (enabling a parallel-processing option or setting up parallel batch processes), make sure you

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.

Figure 10 Transaction STAT: The "Select Statistical Records" Screen



(or your development team) have eliminated the "three Cs" as sources of troubled runtimes:

- A poor **Construct** of the business process, or a mismatch of the construct to a solution
- Inefficient choices for **Customized** settings, which can waste system resources
- Performance traps in the **Code**, such as missing indexes or excessive SQL statements

✓ If the culprit is not found among the three Cs, then the way work items are being distributed for processing may be the reason your high-throughput application is ailing. If it offers a "Parallel processing" checkbox, you may be able to improve throughput by enabling this option.

✓ As we've shown, it is possible for an authorized user to optimize or tune the environment in which a parallel program runs (via a server group definition). In general, however, those unfamiliar with the details of the RFC administration in SAP R/3 should limit their adjustments to the parameters that control the available number of free work processes.

✓ Even if the application you are using does not offer a parallel-processing option, you can set up

parallel batch processes, across which you can distribute the application's load in order to achieve processing efficiencies.

✓ Once your application is running in a parallel-processing mode, if you are not getting expected gains in cycle time, you can work with the database administrator to check for the common culprits:

- Locking situations
- Resource bottlenecks, such as insufficient CPUs or available processes

Susanne Janssen joined SAP in 1997. She is currently a member of SAP AG's Performance & Benchmark Group, a team consisting of performance specialists who consult and support SAP colleagues and customers on mySAP.com performance and scalability. Her responsibilities include information rollout for performance, benchmarks, and sizing. Susanne is also co-developer of the Quick Sizer, SAP's web-based sizing tool. She can be reached at susanne.janssen@sap.com.

Werner Schwarz joined SAP Retail Solutions in October 1998 after working as a developer at two other IT companies. Today, he is the development team contact for all performance-related issues concerning the IBS-CI (Industry Business Sector - Consumer Industries). Werner's major tasks include information rollout and support of his colleagues regarding performance during both development and maintenance. He can be reached at werner.schwarz@sap.com.

 ©2002 SAP Professional Journal. Reproduction prohibited. All rights reserved.