

## Selecting the Number of Strings for VSAM File Under CICS/TS

October 16, 2004

By: Eugene S. Hudders

### CICS/TS Performance Tuning Using C\TREK Course

This seminar covers a lot of tuning material and will require 8 hours a day. The total time for the seminar is 40 hours.

The followings topics included in this course are:

- Introduction to CICS Performance Tuning
- Using Operating System Information to Tune CICS/TS
- Tuning CICS/TS Processor Cycles
- Tuning Real Storage in CICS/TS
- Tuning Virtual Storage in CICS/TS
- Tuning CICS/TS Transaction Controls
- Tuning On Line VSAM Files
- Tuning NSR Files in CICS/TS
- Tuning CICS/TS LSR Buffer
- Tuning the Index CISZ
- Reviewing VSAM Free Space
- Reviewing the DB2 Interface

If you wish to attend this course, please send us an email at [ctrek@actpr.com](mailto:ctrek@actpr.com)

The correct allocation of strings to files is an important tuning opportunity. Under allocating the number of strings assigned can result in short on string conditions that adversely affect response times and could also delay transactions while other resources, such as CPU and virtual/real storage exist. Over allocating strings can result in wasted storage for NSR files because each string has to have a data buffer and in the case of a KSDS, you would also need to have an index buffer. The unneeded strings result in wasted storage that could have been used to improve the performance of another file or LSR pool. This article discusses string assignments not only to individual files but to the LSR pool too.

First of all, let us define what a string is and what its purpose is. A string is nothing more than an access path to a file. It represents how many concurrent accesses we are going to allow to a particular file. A string allows the access method to move the I/O request down to the supervisor level. If the disk is available, the I/O request continues. However, if the disk is busy, then the operating system will queue the request on a device queue. Once the previous I/O completes, the next request on the queue, if any, is selected for execution. Note that the queue is at a very low level in the supervisor. If there are insufficient strings available, CICS queues the request at a much higher level, the address space level. Theoretically, another request for this device from another address space could be placed on the device queue ahead of this CICS request because CICS queuing for string waits occur at a higher level, the address space versus in the operating system. Specifying the use of I/O priority queuing in the z/OS Workload Manager (WLM) could reduce the interference caused by lower priority address spaces. A lower priority request could still be dispatched

because the supervisor is not aware of pending requests that are queued at the address space level. Therefore, strings provide CICS file requests to wait for service in queues at the supervisor level.

Strings are usually over allocated in many installations. There are many reasons for this occurring but in many cases they can be attributed to a lack of understanding of how strings work. Whenever a user application request occurs (e.g., EXEC CICS READ FILE), a string is assigned. The length of time that this string is held depends on how long it takes to complete the request. For example, imagine a device that on the average is reading a record in approximately 6 ms. If the file has three levels of indices, it will require four reads (three for the index and one for the data) for the request to complete. So, if the one read operation took on the average 6 ms., then the request would take 24-ms. to complete. The string would be held for this amount of time. There other requests that would hold the string beyond the end of the I/O such as a read for update or a browse request. In these cases, the string would be held until the rewrite (or delete or unlock) occurred or the ENDBR was issued. Therefore, if strings are held for long periods of time, then there could be a point where you could run out of strings and be waiting on strings. As a result, transactions would have to wait on strings.

There are three possible solutions to wait on strings but in general most people chose the obvious one that is simply increasing the number of strings. Increasing the number of strings also increases the number of assigned buffers as each string requires a data and if applicable, an index buffer (NSR). This could result in a high increase of virtual storage. A second alternative is simply not to do anything. If the number of wait on strings is relatively small (e.g., less than 1%), then you could possibly tolerate the small overhead. However, with the increased virtual and real storage availability, you may not want to tolerate any string waits. The correct solution however, is to reduce the amount of time the string was held. Ensuring that the file is meeting the installation look-aside hit ratios and has proper buffering assigned does this. For example, imagine that in the previous example where we had to read four times for a total of 24 ms. could be done without any physical reads. The time the string would be held would be the instruction time to locate the four records in the buffer that is a much lower value than 24 ms. The string would be released quickly and available for another request. Even if you did not achieve a 100% look-aside hit ratio, you could still improve the total time by finding one to three records in the buffer. Therefore, the correct tuning recommendation is to ensure that the file has proper buffering and is achieving the look-aside hit ratios before increasing the number of strings.

There are some instances where an NSR file has a usually high number of strings allocated. You can observe this condition when reviewing files that have many strings assigned yet the I/O activity against the file is less than other files that reflect higher I/O activity. In many cases, this type of file has many strings allocated because transactions may require more than one string per I/O operation. An example of this condition is a file that issues a read for update request in the middle of a browse without first ending the browse. In this case, a string is maintained for the browse operation and a new string is assigned for the read for update operation. This type of file is prone to string wait conditions that may result in lockouts when being defined with a few number of strings. In many cases the system programmer does not know how many concurrent requests could occur so, the number of strings assigned to the file is over allocated to

compensate for string waits or avoid lockout conditions. This type of programming results in the file having to be defined using NSR because this sequence of I/O requests would result in errors under LSR. The correct solution is to end the browse before issuing the read for update and have the file transferred to LSR support to receive better look-aside capacity.

The same look-aside hit ratio concept applies to string allocations for the LSR pool. In this case, you want to ensure that you are achieving a very high hit ratio on both the index (e.g., 95% or greater) and data (e.g., 80% or greater) buffers in order to reduce the amount of time the string is busy with a request. If after you have properly tuned the LSR buffers and you continue to receive wait on strings, then you should increase the number of strings in the pool. You may however, receive wait on strings at a file level. This occurs when the file has more strings allocated to it than the entire pool has assigned or the sum of the strings required by each individual file exceeds the total number of strings available in the pool. The first condition is an unusual situation and the solution requires increasing the number of strings in the LSR pool to match or exceed the number of strings assigned to the file. The second condition is normal in that the sum of the strings requested for all the files in the pool exceeds the total number of strings assigned to the pool. It should be noted that the pool strings are limited to 255. Therefore, an LSR pool supporting a large number of VSAM files will probably have fewer strings assigned than the sum of the strings assigned to the files. It should be noted that the same conditions could also exist for the buffers allocated.

The measurement of how well the assigned LSR strings are working is measured by the peak number of strings used versus the total number of strings assigned to the pool. The result can be measured as a percentage of peak strings divided by pool strings times 100. The recommended objective for LSR strings is to have the peak number of strings used fall into the 40 to 60% range of the total number of strings assigned to the pool (Figure 1). The difference to 100% is a buffer to cover for unusual increase in use and/or growth. For example, if the peak number of strings used is ten (10), then the defined number of strings for the pool should fall in the range of around sixteen (16) to twenty-five (25). Excessive allocation of strings in an LSR pool results in unused allocated resources, such as virtual storage that could be used to increase the number of buffers in the data and/or index components for improved look-aside hit ratio.

```

_LSRs BANCO POPULAR DE PR          C\TREK ON-LINE          DATE 03/26/2002
APPLID CICSPRD                     APPLICATION DOMAIN      TIME 17:03:42
VERSION 4.1                         LSR STRINGS RECOMMENDATIONS  TERM M178

```

POOL	NUM-OF		CONC-ACT		RECOMM.	
	STRINGS	STRINGS	PERCENT	STRINGS	PERCENT	
1	60	20	33.33	54	37.03	
2	0	0	0.00			
3	18	10	55.55	OK		
4	0	0	0.00			
5	0	0	0.00			
6	0	0	0.00			
7	0	0	0.00			
8	0	0	0.00			

ENTER REFRESH PF1 HELP PF3 PREV PAGE CLR MAIN MENU

MA a 01/001

Figure 1. LSR Pool String Recommendations

The cost for defining unused strings in LSR is not as high as with NSR because strings do not have a data and/or index buffer associated directly with them. So, the virtual storage cost is not as high for LSR as for NSR. However, there are some extra control blocks that are created such as the RPL and PLH that will not be used. In addition, coding the pool key length at 255 bytes will also generate unused virtual storage associated with the unused strings. The unused virtual storage could be put to better use by defining additional buffers that could help increase the look-aside ratio.

Finally, we have reviewed the tuning of strings regarding the use of KSDS files. However, there are some special considerations for string tuning of an on-line ESDS file that is used mainly as a log file. This type of file is used to log records sequentially that can be used as a history or activity file. This type of file is generally characterized by the number of exclusive control waits that occur during the normal operation of creating the log. As this is an output file, VSAM must write the record out to the disk on every request and holds the CI in exclusive control until the operation is complete so as to ensure the file integrity. Due to the high activity usually associated with this type of file, users have a tendency of allocating multiple strings to the file. Since the CI is held in exclusive control, adding additional strings actually results in an increased CPU utilization for the file.

The availability of multiple strings allows CICS to issue the request to VSAM when an outstanding request is in progress. Once this request is received, VSAM will return an exclusive control conflict and will return the request to CICS with the error information. CICS will then

queue the request until the initial request completes. So, the operation used CPU cycles by going to VSAM only to be returned to CICS. However, VSAM string checks are done at the CICS level. So, if the ESDS only has one string assigned, then when the second request is received by CICS, the task is queued due to lack of strings without having to go to VSAM, thus saving CPU cycles. The trade off is changing the exclusive control waits for string waits. There should be very little or no difference in response time. Having extra strings assigned would only be effective when the next request is for the following CI. Thus, if you have a log file that writes one logical record per CI, then extra strings could provide some improvement. However, this may require excessive amount of disk space in the case of small records because disk technology is very sensitive to physical record sizes. So, in general terms, it is best to only allocate one string for ESDS files that are used for output log files.

Tuning Temporary Storage (TS-DFHTEMP) and Transient Data (TD-DFHINTRA) follows the same logic. You need to reduce the number of I/O operations that these files do every day. This is usually accomplished by increasing the number of buffers allocated. However, the more recoverable resources are defined to TS and TD, the more I/O is required due to recovery requirements. So, tuning strings for both of these files follows the same recommendations previously discussed. This completes this article regarding the tuning of strings for VSAM files and the LSR pools.

If you have any questions, comments or request for more information, please contact us:

Address: C\TREK Corporation PO BOX 560069 MONTVERDE FL 34756	Phone: (407) 469-3600 C\TREK Corporation (787) 756-5620 Advanced Computer Technology
E-mail: <a href="mailto:ctrek@actpr.com">ctrek@actpr.com</a>	Fax: (787) 756-5150
Website: <a href="http://www.ctrekcorp.com">www.ctrekcorp.com</a>	Support: (787) 397-4150 (321) 297-5838 (787) 462-0406