



# CICS TS Performance Tutorial -- Other Tuning Opportunities

Eugene S. Hudders  
C\TREK Corporation  
[eshudders@aol.com](mailto:eshudders@aol.com)  
407-469-3600

Session 8269  
March 2, 2011  
8:00 AM

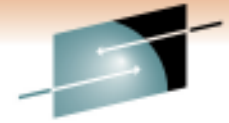


## DISCLAIMERS/TRADEMARKS

- YMMV
- Remember the Political Factor
- CICS/VS, CICS/MVS, CICS/ESA, CICS TS, COBOL LE, COBOL 2, VSAM, DB2, OS/390, MVS, z/OS and z/VSE Are Trademarks of the International Business Machines Armonk, NY

# Agenda

- Introduction Data Tables
  - CICS Maintained Data Tables (CMDT)
  - User Maintained Data Tables (UMDT)
- Temporary Storage
- Virtual Storage
- Real Storage
- Closing



**SHARE**  
Technology - Connections - Results

# Data Tables

**SHARE**  
in Anaheim  
2011

# What is a Data Table?

- With the increased availability of real storage, consideration should be given to more use of Shared Data Tables (SDT)
- A data table is simply a VSAM KSDS that has been placed in VS
  - Entire data set
  - Partial data set
  - Subset of the data set
- SDTs can be shared in the same z/OS image
- Major advantage is that if the desired record is found in the table, no I/O operation occurs
  - Fast access for hits– Use of z/OS Cross Memory Services whenever possible
    - Cross memory services for same image
    - Function shipping may occur
      - *Different z/OS images*
      - *Update requests*
  - Limited I/O (e.g., GET UPDATE/REWRITE)
  - Faster than having the entire file in LSR
    - Data Table uses an in-storage index
    - LSR uses an Index search

# What is a Data Table?

- Candidates for data tables are:
  - Usually small to medium sized data sets
  - Very high read-only access (90% or more read-only operations)
- Support for VSAM KSDS
  - No Alternate Index Support for SDT
    - Changes made to source KSDS via AIX are reflected in the SDT
  - Must be defined to use LSR
- SDTs use data spaces for data and index storage
  - SDT can be greater than 2GB in size
- There are two types of data tables available:
  - CICS Maintained Data Tables (CMDT)
  - User Maintained Data Table (UMDT)
- Global Exits available to tailor (UMDT only) and/or eliminate (or select) records loaded into the table

## CICS Maintained Data Table (CMDT)

- Updates to the CMDT are reflected in the source VSAM data set (as well as the in-storage SDT)
- Full API Support – no application programming changes required
- Variable or fixed length support
- Maximum record length 32 KB
- Easy to implement
- Full recovery is available
- Only non-RLS files supported

## CICS Maintained Data Table (CMDT)

- Commands that access the CMDT
  - READ
    - No UPDATE
    - No RBA option
  - STARTBR, RESETBR, READNEXT and READPREV
    - No RBA option
  - ENDBR (unless sequence has accessed the source data set)
- Commands that access the source data set
  - READ
    - UPDATE
    - RBA option
  - STARTBR, RESETBR, READNEXT and READPREV
    - RBA option
  - ENDBR for sequence that accessed the source data set
  - READ, READNEXT and READPREV for records that are currently processed by a DELETE, REWRITE or WRITE
  - WRITE, REWRITE and DELETE commands



## User Maintained Data Table (UMDT)

- Updates to the UMDT are NOT reflected in the source VSAM data set (only the SDT is updated)
  - Source data set is closed immediately after load is completed
  - Source is independent of the UMDT
  - Any update activity would have to be handled by the user
- Must specify variable length records
- Maximum record length is 32 KB
- Some API changes are required for the application (Limited API support)
- Easy to implement

# Data Spaces (DS)

- Data is stored in several data spaces whether or not the SDT is shared or not
- Maximum support for 100 data spaces
  - Each data space can be up to 2 GB in size (Max 200 GB)
  - DS size can be controlled by installation IEFUSI exit
- The initial data spaces used are:
  - DFHDT001 – used for table entry descriptors
  - DFHDT002 – used for index nodes
  - DFHDT003 – used for storing the data records
  - DFHDT004 to DFHDT100 – available for data record space expansion (if required)
- Data Spaces are shared by all SDTs used by the CICS region
- Data tables remain until CICS region is brought down
  - Storage used by a data table is released if the data table is closed
  - Released storage is available for reuse

## Data Spaces (DS)

- Space within the DS is allocated in 16 MB increments
  - Sub-allocation in smaller quantities occur for the different requirements:
    - 32 KB for table entry descriptors
    - 32 KB for index nodes
    - 128 KB for data records
  - New storage increments are allocated as required (16 MB increments)
  - If maximum DS size reached, a new additional DS is allocated
    - If maximum capacity reached, CICS notes DS is full
    - New requests for space fail

## Data Spaces (DS)

- There is no facility for viewing amount of space allocated
  - CICS statistics provide information as to how much storage is allocated and used by file

## Global User Exits

- XDTRD – select records that can be loaded into the SDT when the file is initially read and the table created
  - For UMDT, this exit can be used to modify the records loaded into the table
- XDTAD – select records to be added to the table when records are added to the data set
- XDTLC – perform some processing at the end of the SDT load

## Defining a Data Table

- RDO File definition parameters:
  - TABLE (NO|CICS|USER|CF)
  - MAXNUMRECS (NOLIMIT|number)
    - Maximum # of records supported is 16,777,215
  - FILE (name)
  - DSN (name)
  - LSRPOOL (number|1)
    - Any I/O operations required by the SDT are done using LSR

## SDT Candidates

- Consider using CMDT because:
  - Are easier to implement
  - Have a more complete API precluding application program changes
  - Data sets that have a high amount of read-only activity (90%)
  - Small to intermediate sized data set
  - High payback for data sets that are accessed by remote regions
    - Function shipping overhead versus cross memory services

## SDT Candidates

- Consider using UMDT for:
  - Data sets that have update activity but do not have to update source file
  - The amount of information required is smaller than the record size (subset)
  - Require information from other sources that are not a VSAM KSDS (RLS, IMS, DB2 etc.)



## SDT Performance Issues

- SDT can provide better performance for read-only files than LSR
  - CPU utilization
    - Search for record requires less CPU time than with LSR
    - Initial load of the data table uses LSR resources
    - Write operations require the use of LSR

# SDT Performance Issues

- Storage utilization
  - Efficient use of CICS Region storage because information is kept in a data space
    - *Data records are stored in DFHDT003 (and up)*
      - *Storage in this data space is allocated in 16 MB increments*
      - *Record storage is allocated in 128 KB increments*
      - *Records are stored in page-aligned frames that loosely resemble the CI*
      - *Where ever possible, records are stored next to records with closest lower key*
      - *If many records are added and/or record lengths increased, then records are added randomly across the data space with an increased amount of storage (e.g., 2X)*

## SDT Performance Issues

- *Table-entry descriptor storage are allocated in storage acquired from data space DFHDT001*
  - *Space is allocated in increments of 32 KB*
  - *There is one table-entry descriptor per record in the data set **PLUS***
  - *One table-descriptor entry for each gap in the key sequence (e.g., where one or more records have been omitted from a CMDT)*
  - *The size of each entry is  $KL + 9$  bytes rounded to a double word boundary*

## SDT Performance Issues

- *Index node entries are allocated in storage acquired from data space DFHDT002*
  - *Space is allocated in increments of 32 KB*
  - *The size depends on number of records as well as the format of the key values and distribution of the keys*
  - *In the case of dense keys (all keys are consecutive – no gaps) – Binary = 5.1 bytes; Decimal = 8.5 bytes; Alphabetic = 19 bytes*
  - *In the case of sparse keys (no keys are consecutive – gaps exist) – decimal = 44 bytes; Alphabetic = 51 bytes*
  - *Worst case scenario = 76 bytes*

# SDT Performance Issues

- *Some ECSA storage is required for communication with other regions that share the SDT*
  - *Used for control block storage*
- *Bottom line is that converting from LSR to an SDT may lead to an increased use of real storage in exchange for lower CPU utilization*
  - *Possible alternatives are reducing number of LSR buffers, if file was converted from LSR*
  - *Eliminating read-only files that were replicated among regions*
- *Deleted space within an SDT remains available for that particular SDT until the data set is closed*
- *Free space within the data table is tracked and reused where possible*
- *Free space within a frame are not necessarily in sequence and consolidation of free space is not done – records are located indirectly by descriptors*
- *Consolidation would preclude concurrent reading by other regions*

# SDT Performance Issues

- From where does the magic **90%** read-only ROT for data tables come?
- Any output related requests (READ for UPDATE, WRITE, etc.) are automatically directed to the source VSAM KSDS (data table is also updated)
- Lets review:
  - VSAM KSDS
  - 100,000 requests
  - 90% read operations
  - Breakdown
    - 90,000 Reads
    - 10,000 Output related requests (READ for UPDATE/REWRITE/DELETE)

# SDT Performance Issues

- Worst case scenario if you have to go to disk for the data:
  - 1 Index level data set – 2 I/O operations required (1 for the index and 1 for the data)
  - 2 Index level data set – 3 I/O operations required (2 for the index and 1 for the data)
  - 3 Index level data set – 4 I/O operations required (3 for the index and 1 for the data)
- So, with 10,000 read for update operations
  - 1 IX = 20,000 I/O operations
  - 2 IX = 30,000 I/O operations
  - 3 IX = 40,000 I/O operations
- The figures do not consider the output I/O
  - REWRITE/DELETE/UNLOCK
- Also, we are not considering any I/O as a result of adds (insertions) to the file and any subsequent I/O caused by CI/CA Splits

## SDT Performance Issues

- It is important to note that any output operation to a CMDT will entail I/O
- A CMDT data set has to be assigned to LSR
- Therefore, there are two types of look-aside hits possible for a CMDT:
  - A look-aside hit at the table level
  - A look-aside hit at the LSR pool level
- The difference is the amount of CPU required to locate the record



## SDT Performance Issues

- LOOK-ASIDE FORMULA:

- Look-Aside % =  $\frac{\text{Hits}}{\text{Hits} + \text{I/O Operations}} * 100$

Note: Hits includes direct hits to the data table plus CIs found in the LSR pool

## SDT Performance Issues

- So, using a worst case scenario with **0%** look-aside in LSR, the data table look-aside hit ratios are:
  - 1 IX =  $((90000/(90000+20000)*100)) = 82\%$
  - 2 IX =  $((90000/(90000+30000)*100)) = 75\%$
  - 3 IX =  $((90000/(90000+40000)*100)) = 69\%$
- In this case, there were no CIs found in the LSR pool requiring an I/O operation to occur

## SDT Performance Issues

- Then, using a slightly higher case scenario with a 50% look-aside in LSR, the data table look-aside hit ratios are:
  - 1 IX =  $((95000/(95000+(20000*.5))*100)) = 90\%$
  - 2 IX =  $((95000/(95000+(30000*.5))*100)) = 86\%$
  - 3 IX =  $((95000/(95000+(40000*.5))*100)) = 83\%$
- As we indicated that there would be a 50% look-aside hit ratio in the LSR pool, then the number of “look-aside” hits for the data set has to be adjusted by those CIs found in the LSR buffers
  - The difference is slightly more CPU usage

## SDT Performance Issues

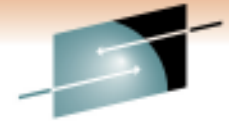
- Finally, using a better case scenario with 90% look-aside in LSR, the data table look-aside hit ratios are:
  - 1 IX =  $((99000/(99000+(20000*.1))*100)) = 98\%$
  - 2 IX =  $((99000/(99000+(30000*.1))*100)) = 97\%$
  - 3 IX =  $((99000/(99000+(40000*.1))*100)) = 96\%$
- The higher LSR look-aside hit ratio greatly improves the overall hit ratio for the data set

## SDT Performance Issues Recommendations

- The trick to good performance with data tables that receive some type of write commands is to ensure a good LSR hit ratio
  - This may be difficult if the table shares a pool with other non-data table data sets because other more active data sets in the pool may push the data table's CIs out of the buffer pool
  - **Therefore**, use a separate LSR pool for data tables where any contention only comes from other data table files
    - Ensure excellent index buffering
- Data tables that receive insertions (adds) can cause additional I/O and a significant increase in virtual/real storage requirements (not recommended)

## SDT Performance Issues Recommendations

- In the case of File Owning regions (FOR) Shared Data Tables (SDT) should be used for read-only data sets that are propagated across regions to avoid function shipping requests
- Data tables that receive insertions (adds) can cause additional I/O and a significant increase in virtual/real storage requirements (not recommended)



**SHARE**  
Technology - Connections - Results

# Temporary Storage

**SHARE**  
in Anaheim  
2011

# TEMPORARY STORAGE (TS)

- Two types of Temporary Storage (TS) are available:
  - TS MAIN – queues are maintained in virtual storage (ECDSA)
  - TS AUX – queues are written to auxiliary storage (disk)
    - Recovery is possible for TS Aux
    - Disk data set is DFHTEMP
      - *VSAM ESDS data set*
      - *Formatted and handled by TS*
- Major problem with TS today is that many programmers forget that this facility is for TEMPORARY use
  - TS = Permanent Storage!



# TS MAIN

- To implement TS MAIN you should ensure that:
  - You have sufficient VS expansion room within the REGION
  - You have sufficient real storage and are not paging
- You can force all requests to TS MAIN by specifying no buffers and strings in the TS SIT parameter
  - You can also use a TS Model
- Tuning TS MAIN is mainly ensuring:
  - Sufficient EDSA is available
  - A large REGION specification to accommodate a larger EDSALIM
- There are several ways to determine the EDSA requirements for TS MAIN
  - Study the Peak Storage used in DFHTEMP for several days to determine the HWM
    - Using the peak CI used value times the CISZ gives a starting value for VS
    - Add the current TS MAIN peak usage
    - Using the previous figure increase by a % or growth factor
    - As this is usually a "guess-timate", you should occasionally monitor the TS MAIN use
  - Use the total CIs allocated for DFHTEMP times the CISZ
- Major advantage of TS MAIN is speed of access to queues
- Major disadvantage is that you are exposed to SOS
  - Programming guidelines and enforcement regarding queue deletion is required
  - A generic TS queue delete program/exit may be safety valve

# TEMPORARY STORAGE




```

DALLAS03 - EXTRA X frame
File Edit View Tools Session Options Help
-----
Display Filter View Print Options Help
-----
SDSF OUTPUT DISPLAY CICSSTS32 STC06825 DSID 122 LINE 4,340 COLUMNS 01- 80
COMMAND INPUT ==> _ SCROLL ==> CSR
-Temporary Storage -
+-----+
0 Put/Putq main storage requests . . . . . 0
  Get/Getq main storage requests . . . . . 2
  Peak storage used for TS Main. . . . . 0K
  Current storage used for TS Main . . . . . 0K
                                     |
Put/Putq auxiliary storage requests. . . . . 1,526
Get/Getq auxiliary storage requests. . . . . 0
                                     |
Times temporary storage queue created. . . . . 1,518
Peak temporary storage queues in use . . . . . 7
Current temporary storage queues in use. . . . . 7
Items in longest queue . . . . . 9
0 Control interval size. . . . . 4,096
  Control intervals in the DFHTEMP dataset : 359
  Peak control intervals used. . . . . 29
                                     |
F1=HELP      F2=SPLIT      F3=END      F4=RETURN      F5=IFIND      F6=BOOK
F7=UP        F8=DOWN       F9=SWAP     F10=LEFT       F11=RIGHT     F12=RETRIEVE
-----
4B          :00.1          04/21
-----

```

**TS MAIN USAGE STATS**

**CISZ, TOTAL CIs in DFHTEMP and PEAK CIs USED**

# TS AUX

- Most tuning to TS AUX is related to the DFHTEMP data set
- The major tuning item is to eliminate I/O to DFHTEMP
  - I/O reduction is a function of adding buffers
  - TS uses delayed writes – that is, the buffer is not written to DFHTEMP until the buffer is needed
    - So, if the READQ finds the queue in a buffer, then an I/O operation is eliminated
  - **I/O Look-Aside Hit Ratio =  $(1 - ((\text{Buffer Reads} + \text{Buffer Writes}) / (\text{PUTQ Aux} + \text{GETQ Aux})) * 100)$** 
    - Objective would be to achieve 80% or greater look-aside hit ratio
  - So, with a smaller VS investment than TS MAIN, a user can achieve a “simulated TS MAIN” with acceptable response time
  - However, there is a price attached – more buffer compressions increase CPU usage
    - Buffer compressions are still better than an I/O
- TS can handle records greater than the CISZ assigned to DFHTEMP
  - Writes greater than CISZ add CPU overhead
    - IF  $((\# \text{ OF WRITES GT CISZ} / \# \text{ OF AUX WRITES}) * 100)$  EXCEED 3%, THE DFHTEMP CISZ SHOULD BE INCREASED
    - NEW CISZ SHOULD NOT BE BASED ON THE LONGEST RECORD WRITTEN TO DFHTEMP
    - BE SURE TO ADJUST DISK SPACE ALLOCATION TO ENSURE SAME NUMBER OF AVAILABLE CIs IN DFHTEMP TO AVOID SHORT ON AUX CONDITIONS

# TS AUX

- TS format writes indicate that DFHTEMP had to be extended
  - The primary allocation should be sufficiently large so that the peak CI usage is in the 60-70% range of the primary allocation
  - Use secondary allocation as a safety valve
  - Format writes should not occur
  - Increase the primary allocation for DFHTEMP to include any secondary allocations for the DFHTEMP reorganization
- Increasing the # of strings to resolve string wait conditions on DFHTEMP may not be the appropriate solution
  - Evaluate if you are achieving the proper look-aside hit ratio (e.g., 80%)
    - If NOT, then add more buffers
    - If yes, then add more strings
- The major advantage of TS AUX is that you can achieve good response times with proper buffering without committing a lot of VS
  - TS Aux full versus SOS – which is worst?
- The major disadvantage is that it is not as fast as TS MAIN

# TEMPORARY STORAGE

```

DALLAS03 - EXTRA X frame
File Edit View Tools Session Options Help
-----
Display Filter View Print Options Help
-----
SDSF OUTPUT DISPLAY CICSSTS32 STC06825 DSID 122 LINE 4,340 COLUMNS 01- 80
COMMAND INPUT ==> _ SCROLL ==> CSR
-Temporary Storage -
+-----+
0 Put/Putq main storage requests . . . . . 0
  Get/Getq main storage requests . . . . . 2
  Peak storage used for TS Main. . . . . 0K
  Current storage used for TS Main . . . . . 0K

  Put/Putq auxiliary storage requests. . . . . 1,526
  Get/Getq auxiliary storage requests. . . . . 0
                                     TS AUX STATS

  Times temporary storage queue created. . . . . 1,518
  Peak temporary storage queues in use . . . . . 7
  Current temporary storage queues in use. . . . . 7
  Items in longest queue . . . . . 9
0 Control interval size. . . . . 4,096
  Control intervals in the DFHTEMP dataset : 359
  Peak control intervals used. . . . . 29

F1=HELP      F2=SPLIT      F3=END      F4=RETURN      F5=IFIND      F6=BOOK
F7=UP        F8=DOWN       F9=SWAP     F10=LEFT      F11=RIGHT     F12=RETRIEVE

4B  :00.1 04/21

```

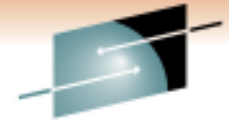












**SHARE**  
Technology - Connections - Results

# Virtual Storage

**SHARE**  
in Anaheim  
2011

# Introduction

- Virtual Storage (VS) has traditionally been the “Achilles Heel” of CICS
- Lack of VS causes the system to be under stress that results in SOS conditions and response time elongation
- CICS/TS design of separate (E) DSAs and dynamic (E) DSA allocation should limit the occurrences of SOS conditions, if appropriate region and limit VS are allocated
- Baring a program loop error, there are *no* valid reasons for running SOS or reaching limit conditions above the line VS
  - Possible exception – JAVA
- Major giveaway that there may be a VS problem within CICS/TS is excess program loading

# Introduction

- Among the new things in the CICS/TS design to help with VS are:
  - Eight different dynamic storage areas
    - Dynamic storage allocation for (E) DSAs
  - Dynamic (E) DSA allocation based on the VS allocated in the EDSALIM/DSALIM SIT parameters
  - Periodic program compression based on a percentage of available free storage
    - Only compress sufficient programs to relieve situation
  - Delay penalty for new transactions when approaching VS limit conditions
  - Better storage violation recovery of storage areas

# Introduction

- Traditional VS tuning opportunities are still valid some of which are:
  - Split system (MRO/ISC)
  - Tune for VS
  - Increase CICS Region size
  - Increase DSALIM/EDSALIM, if possible
- Major obstacle to be faced is getting OS system programmers to allow the setting of REGION=0M in JCL start-up for CICS
  - IEFUSI can be an obstacle

# Introduction

- Tuning for VS is somewhat similar to tuning for Real Storage (RS)
  - Optimize performance
  - Use less resources
  - Eliminate unneeded resources
  - Improve transaction occupancy time (e.g., response time)
- Different from RS, you may not be able to buy more VS

# REGION

- REGION
  - Not a SIT parameter
  - REGION=0K is recommended to allow the user the capacity to tune the CICS system without having to worry (a lot) about the amount of virtual storage available to accommodate the changes (e.g., increase EDSALIM, add LSR buffers etc.)
  - Many installations control Region size via the IEFUSI SMF exit
  - **Can be a political issue**
  - CICS will only allocate/use what it needs

# MEMLIMIT

- MEMLIMIT
  - Not a SIT parameter
  - CICS TS 3.2 needs to have at least a size greater than EDSALIM (warning message if less than 2 GB but GT EDSALIM or if less, CICS will not run)
  - Recommended size is 2 GB or more
  - If REGION=0K and no MEMLIMIT is provided, system takes a default of NOLIMIT
  - Several ways to assign MEMLIMIT but two most common are through the JCL or IEFUSI SMF exit
  - Used for container and control block storage in CICS TS 3.2
  - Watch out for z/OS page data set requirements

# MEMLIMIT Above the Bar

- MEMLIMIT Information – STAT

```

MEMLIMIT Size . . . . . : 200G ←
MEMLIMIT Set By . . . . . : JCL
GETSTOR request size. . . . . : 3,072M
Current Address Space active. . . . . : 2M
Peak Address Space active . . . . . : 2M
Current GDSA Active . . . . . : 2M
Peak GDSA Active. . . . . : 2M ←
Above the bar Cushion Limit . . . . . : 194,560M ←
Allocates into the Cushion. . . . . : 0
                                     GCDSA
-----
Current DSA Size. . . . . : 1M
Peak DSA Size . . . . . : 1M
Getmain Requests. . . . . : 0
Freemain Requests . . . . . : 0
Current number of Subpools. . . . . : 1
Add Subpool Requests. . . . . : 0
  
```



## DSA Limit

- DSALIM
  - Default is 5 MB
  - Used to allocate the DSA below the line
  - Make as large as required to support transactions that run below the line
  - Ideal objective for DSALIM is that Peak Storage used be 70% of the DSALIM requested

# DSA Below the Line

- DSALIM Information – STAT

```

Private Area Region size below 16Mb . . . . . : 9,192K
  Max LSQA/SWA storage allocated below 16Mb (SYS) . . : 380K
  Max User storage allocated below 16Mb (VIRT). . . . : 6,480K
  System Use. . . . . : 20K
  RTM . . . . . : 250K
-----
Private Area storage available below 16Mb . . . . . : 2,062K

VIRT minus Current DSA Limit. . . . . : 336K
-----
MVS PVT Size. . . . . : 9,216K
MVS CSA Size / Allocated. . . . . : 3,748K / 252K
MVS SQA Size / Allocated. . . . . : 1,540K / 202K

Current DSA Limit . . . . . : 6,144K ←
Current Allocation for DSAs . . . : 2,048K
Peak Allocation for DSAs. . . . . : 2,048K ←
  
```

## Extended DSA Limit

- EDSALIM
  - Default is 34 MB
  - Used to allocate the Extended DSA above the line
  - Sometimes under-allocated
    - SOS conditions
    - Program loads
      - *Program compressions uses CPU cycles*
      - *An indication that SOS conditions are “around the corner”*
  - Make as large as required to support transactions that run above the line
  - Ideal objective for EDSALIM is that Peak Storage used be 70% of the EDSALIM requested

# EDSA Above the Line

- EDSALIM Information – STAT

```

Private Area Region size above 16Mb . . . . . : 1,609,728K
  Max LSQA/SWA storage allocated above 16Mb (SYS) . . : 12,880K
  Max User storage allocated above 16Mb (EXT) . . . . : 436,204K
-----
Private Area storage available above 16Mb . . . . . : 1,160,644K

CICS Trace table size . . . . . : 64K
EXT minus Current EDSA Limit. . . . . : 26,604K

MVS EPVT Size . . . . . : 1,609,728K
MVS ECSA Size / Allocated . . . : 400,604K / 29,112K
MVS ESQA Size / Allocated . . . : 14,952K / 9,960K

Requests for MVS storage causing waits . . : 0
Total time waiting for MVS storage . . . : 00:00:00.00000

Current EDSA Limit. . . . . : 409,600K ←
Current Allocation for EDSAs. . . : 83,968K
Peak Allocation for EDSAs . . . : 83,968K ←
  
```

# LOADER DOMAIN

- An early alert that you have you need to correct the DSALIM and/or EDSALIM is provided by the Loader Domain
  - Statistics regarding program loads as a result of being “removed” (greater than zero) indicates that some program compression has occurred to free-up VS
  - Program loads/compression add additional CPU overhead
    - Ensure largest possible DSALIM/EDSALIM allocations
    - Start region with REGION=0M
    - Maximum possible size for DSALIM/EDSALIM
      - *CICS will allocate only what it needs*
- Eliminate unnecessary loads with hold conditions without an intervening release
  - Increases the overhead to process the system LLE list
  - Eats up ECDSA storage for repetitive LOAD with HOLD without intervening release
    - Small entry in terms of bytes but if CICS is not recycled, it can continue to grow

## Deadlock Timeout –DTIMOUT

- Deadlock time out value is used by CICS to purge transactions that have been suspended (e.g., due to SOS)
- Deadlock time is specified in the transaction definition (Max 68 minutes) (MMSS)
- For DTIMOUT to be effective, SPURGE=YES must also be specified
- All non-update transactions should have DTIMOUT specified to allow CICS to take action deadlock conditions such as SOS

## Deadlock Timeout –DTIMOUT

- Recommendation is to specify DTIMOUT and SPURGE for all transactions that do not have any updates to resources such as:
  - Inquiry
  - Menu
  - Browse

## Tuning for VS Above/Below

- Tuning for VS above the line is usually relatively easy – increase the OS Region size, if required, and increase the EDSALIM specification
  - May require a new CICS clone in the case of heavy JAVA usage
- Tuning for VS below the line is not as easy because of the limited space available below the line
  - Can be as high as 9 MB and as low as 5 MB
  - Requires a lot more effort, testing and planning
  - Actions should be done gradually and phased into production
  - Be careful when implementing LPA support for CICS – avoid moving CICS modules into the LPA that are below the line unless there is sufficient VS space without affecting the zOS CSA one MB boundary

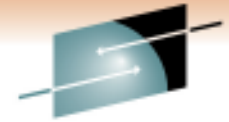


## Tuning for VS Above/Below

- If possible, always leave a 1 MB boundary below the line
  - For example, if 8 MB available, don't make the DSALIM greater than 6 or 7 MB
- Do not use Transaction Isolation (TI) if heavy transaction load below the line
  - UDSA may be affected by fragmentation caused by other DSA allocations (CDSA/RDSA/SDSA)
  - May require the use of the SIT override to allocate a fixed amount of storage to the UDSA (e.g., 3 MB)
  - If high transaction load below the line, consider the use of TCLASS to control transactions below the line instead of MXT

## Tuning for VS Above/Below

- Action items to tune for 24-bit VS
  - Move application code above the line
  - Move TCTUA above the line or lower maximum size to a used value plus a buffer
  - Make sure MAPs are link edited above the line
  - Eliminate unneeded TWAs that are defined for tasks that run below the line
  - Increase DSALIM to maximum without affecting the “above the region” significantly
    - Need about 300 KB for zOS RTM processing
  - Ensure that programs linked AMODE 31 and RMODE ANY do not have a DATA (24) specified in the CBL



**SHARE**  
Technology - Connections - Results

# Real Storage

**SHARE**  
in Anaheim  
2011

# Introduction

- CICS performance is affected by the lack of real storage
- General symptom is an irregular response time
  - Same transaction response time varies greatly
  - Response time usually deteriorates slowly when transaction volume increases
  - Recovery usually observed as transaction volume is reduced
- First area to analyze is the CICS page-in rate
  - Check the overall UIC
  - Check Region page-in rate
  - Check overall system page-in rate
- NOTE: Real storage is not a general problem in today's environment

## CICS TS and Real Storage

- Multiple CICS regions require duplicate working set for the CICS management routines
- Approximately 600 to 800 KB of central storage after the 1<sup>st</sup> CICS can be saved if the CICS management modules are moved to the (E) LPA
  - Better control of real storage than the ICV
  - Requires an IPL
  - Consideration for maintenance must be given

# Operating System and Real Storage

- Care must be taken when moving modules into the LPA (below the line)
  - CSA must be on a 1 MB boundary (Segment)
  - One byte that exceeds the segment boundary, will cause an entire 1 MB loss to CICS (and all regions) below the line
  - Recommendation → do not move CICS below the line modules into the LPA unless you have room to spare

# Closing

- Best tuning option – eliminate I/O
  - Data Tables provides a viable option to reduce/eliminate I/O operations
    - Look-aside for record occurs at both the data table and LSR buffer pool
    - Separate data table files into separate LSR pool
  - TS Aux tuning is mainly oriented to reduction of I/O
    - Assign sufficient data buffers
- Tuning for storage varies
  - Virtual storage (VS) tuning most important below the line
    - Allow sufficient VS at CICS initialization
  - Real storage (RS) tuning may not be as important today due to increased availability of real storage on mainframes
  - Future challenge will be the implementation of storage support above the Bar

## Questions

- Thank-you
  - ANY Questions?