



# **PERFORMANCE TUNING IN A CICS/TS ENVIRONMENT**

**EUGENE S HUDDERS  
CTREK CORP.  
PO BOX 560069  
MONTVERDE FL 34756  
407-469-3600**

# NOTES

- **z/VSE, z/OS, CICS/VS, CICS/MVS, CICS/ESA, CICS/TS, COBOL LE, COBOL 2, VSAM, DB2, OS/390, MVS Are Trademarks of the International Business Machines Armonk, NY**
- **This presentation contains a series of recommendations that have to be tested in the users environment**



# AGENDA

- **INTRODUCTION TO CICS/TS TUNING**
- **TUNING NSR FILES**
- **TUNING LSR POOLS**
- **TRANSACTION CONTROLS AND TUNING CPU RESOURCES**
- **CLOSING**
  - **QUESTIONS AND ANSWERS**



# INTRODUCTION TO CICS/TS TUNING



Copyright © 2007. CTREK Corporation. All Rights Reserved.

# INTRODUCTION

- **CICS continues to be the premier transaction processor**
  - **Over 35 years old and approximately \$1 Trillion invested in on-line applications (IDC)**
  - **Over 950K programmers**
  - **Over 30B transactions processed on a daily basis**
    - **Over \$1 Trillion processed/day**

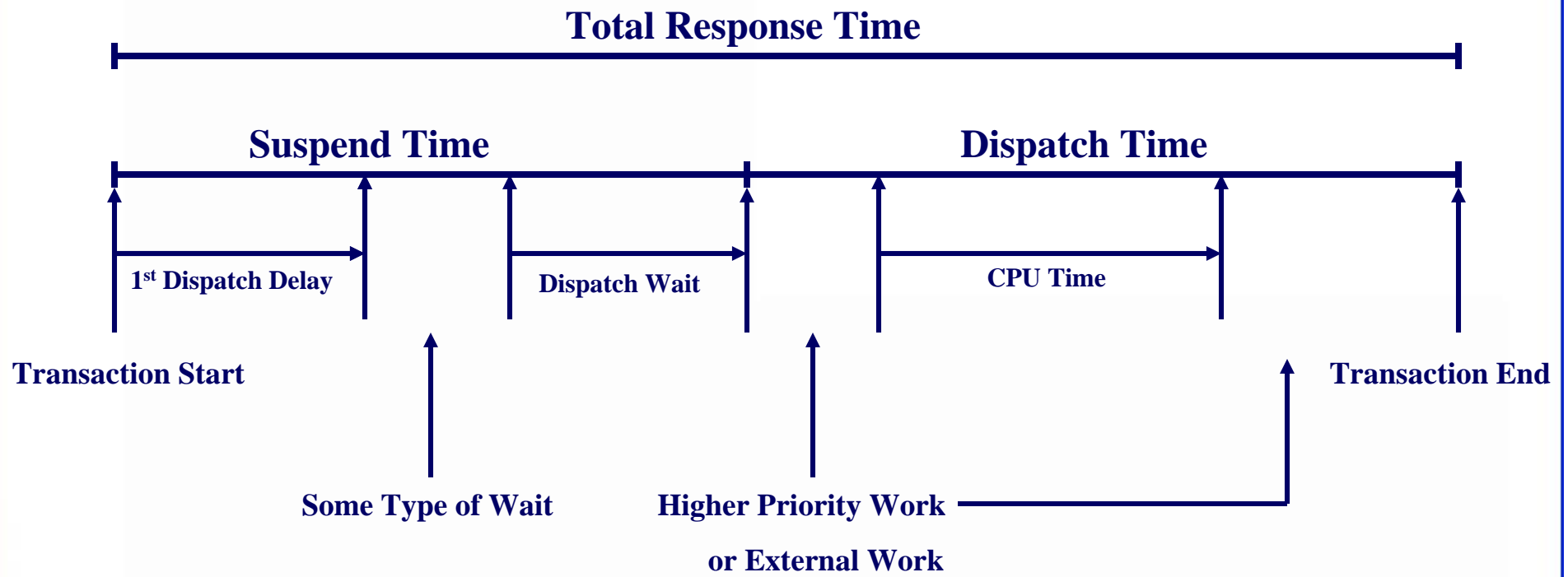


# TUNING

- **The first thing to determine is why are you tuning?**
  - **Performance problem**
  - **Lack of certain resources**
  - **Response time**
  - **Capacity planning – setting the base**
  - **Learning experience**
  - **Avoid hardware upgrades**
  - **Don't have anything better to do**



# Response Time



# Types of Waits

- **There are many types of waits that are attributed to transaction suspend time, such as:**
  - Terminal wait
  - TS/TD wait
  - File waits
    - String waits
    - Buffer waits
    - Exclusive Control Conflict waits
  - Journaling wait
  - Inbound/Outbound Socket wait
  - Inter-region (MRO/ISC) wait
  - Other
    - ENQ wait (Local or Global)
    - Interval Control wait (Time)
    - Lock manager wait
    - External wait
    - CICS Waits (SOS/MXT)
    - Dispatcher delay



# Conditions That Affect Response Time

- **Stress conditions**
  - SOS
  - MXT
  - Looping transactions
  - Other limit conditions (TCLASS)
- **Paging**
- **Program Loading**
- **Storage Violations**



# Conditions That Affect Response Time

- **Hardware conditions**
  - **CPU cycles**
    - LPAR weights
  - **Real Storage**
  - **I/O contention**
  - **Network delays/outages**



# Conditions That Affect Response Time

- **Software conditions**
  - Application programming and design
  - Data base design
  - Network design
    - Web
    - BMS
  - Single threading



# Constraints

- **Hardware**
  - **DASD**
  - **Real Storage**
  - **Paths (Channels)**
  - **Network**
- **Virtual/Real Storage**
  - **Below the Line**
  - **Paging**
- **CPU Cycles**
  - **“CICS Loves Fast Engines”**



# Closing

- **Tuning is something that has to be done to CICS to ensure:**
  - Better response time
  - Better use of available resources
  - Clearing possible bottlenecks that may lead to future problems
- **The major problem with tuning CICS are:**
  - Lack of experienced personnel
  - Lack of resources to dedicate to the task
  - The total number of CICS systems that need to be tuned
  - The level of detail required to properly tune the system
  - The tools available may not be suited for the task



# Tuning CICS NSR Files



Copyright © 2007. CTREK Corporation. All Rights Reserved.

# Introduction

- **CICS uses two buffering techniques to handle VSAM files within CICS/TS**
  - **Non-Shared Resources (NSR)**
  - **Local Shared Resources (LSR)**
- **In recent years, new VSAM features announced for CICS have been LSR oriented**



# Introduction

- **The major difference between the two buffering techniques lies in the “ownership” of the resources**
  - NSR → resources are used exclusively by the file
  - LSR → resources are shared between participating files
- **CA splits tie up the main task KTCB for NSR files**
  - Consider moving the file to LSR



# Introduction

- **LSR advantages include:**
  - More efficient VS use because resources are shared
  - Better look aside because buffers can maintain the sequence set records and look-aside from any string is allowed
  - Tends to be more self-tuning because buffers are allocated on an LRU basis keeping information of the more active files in the buffers at the expense of less active files
  - Only one copy of a CI allowed (better read integrity)
  - CI/CA splits do not cause the main subtask to wait
  - Can allocate up to 15 pools to segregate important files



# Introduction

- **NSR advantages include:**
  - Resources are reserved so one file can be specifically tuned
  - Allows for chained read operations that can give better sequential performance
    - BROWSE
    - CA Splits
    - Mass inserts



# Introduction

- **IBM's recommendations are to use LSR except:**
  - A very large, active file that has little opportunity for look aside
  - Files that have sequential operations such as browse operations, mass inserts or CA splits
- **We will analyze NSR from a KSDS point of view**



# NSR File Definition

- **String definition for an NSR file can be a challenging task**
  - Many NSR files are over allocated in strings when considering the I/O activity against the file
  - The major reason is that NSR allows duplicate CIs to exist between strings
  - NSR allows STARTBR/READNEXT/READ for UPDATE sequence without an intervening ENDBR
    - This results in two strings being allocated to the task
    - The requested CIs appear twice in VS
    - As a result, many files would appear to be deadlocked due to lack of strings
    - This type of request will not work in LSR
  - Remember that a string needs a BUFND/BUFNI
    - Eliminate strings in favor of more buffers



# NSR File Definition

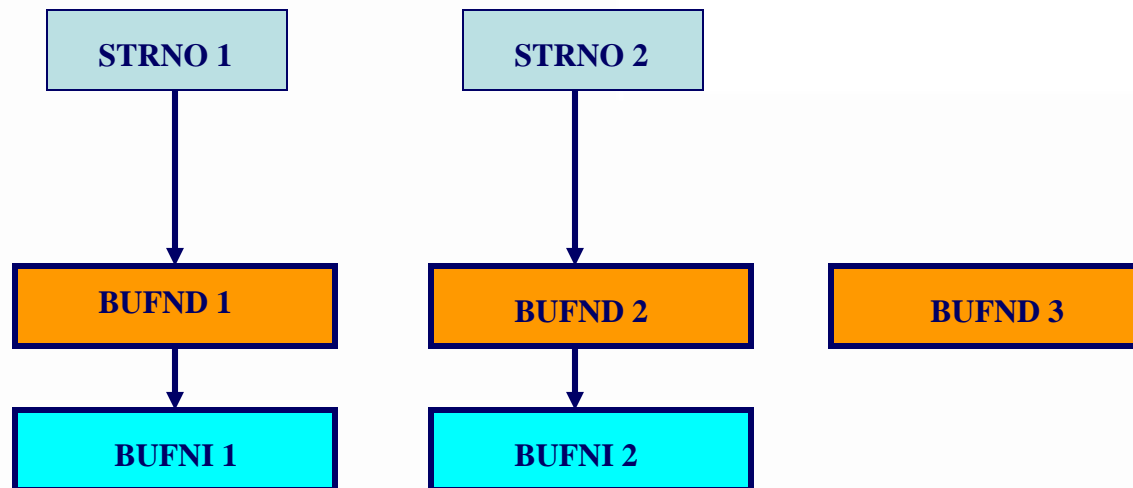
- **Additional buffers can be allocated**
  - **Extra BUFND** – will be used in sequential operations
    - Half of available buffers will be allocated to the 1<sup>st</sup> sequential request
  - **Extra BUFNI** – will be used to store index set indices (high level indices)
  - **Sequence Set Indices (SSI)** are never read into the extra BUFNIs
    - SSI CIs are read into the string index buffer
    - No look aside to other string buffers are done



# NSR Buffer Definition

- **Example # 1:**

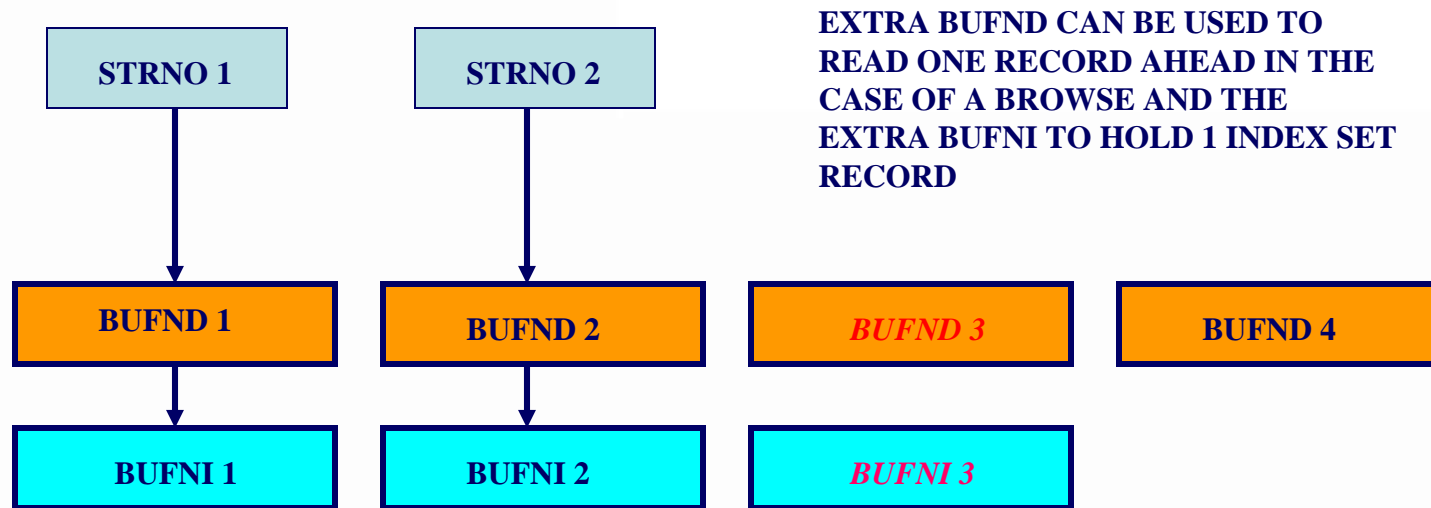
– STRNO = 2      BUFND = 3      BUFNI = 2



# NSR Buffer Definition

- **Example # 2:**

– STRNO = 2      BUFND = 4      BUFNI = 3



EXTRA BUFND CAN BE USED TO  
READ ONE RECORD AHEAD IN THE  
CASE OF A BROWSE AND THE  
EXTRA BUFNI TO HOLD 1 INDEX SET  
RECORD



# NSR Performance

- **Why would you want extra BUFNDs?**
  - In the case of a BROWSE request, to read ahead a number of CIs to improve performance of the task
    - At the expense of the rest of the system and possibly other tasks accessing this file
  - In the case of Mass Insert, to write behind a series of CIs to improve task performance
  - In the case of CA Splits, to be able to move more than one CI at a time to the new CA
  - Overall, extra data buffers can speed up the process and reduce I/O requests to the file



# NSR Performance

- **What is the hidden agenda?**
  - **Browse**
    - The number of BUFNDs defined should contain the approximately the same number of records read (READNEXT) by the program
      - For example, if a CI can contain 5 records and the average # of READNEXT operations issued is 20, then a BUFND specifying 4 additional buffers (5 records/CI\*4 read ahead buffers) would be fine
      - However, what programmer knows on the average how many READNEXT operations are issued to a file?
      - Also, usually only the 1<sup>st</sup> BROWSE request would benefit
      - What happens if the BROWSE is ended (ENDBR) before the 20 READNEXT operations are done?
    - Adding additional buffers for sequential BROWSE processing will increase the task response time because of an elongated I/O operations will occur
    - In addition, having the data in storage is good for this task but may affect the response of other tasks in the system
      - Lockout of the QR KTCB



# NSR Performance

## – Mass Inserts

- The number of buffers should be around the same number of writes (WRITE) issued to the file at one time
  - Same logic as the BROWSE
- However, if the number of writes ends before all the buffers are full, then there is no I/O penalty as in the case of a BROWSE

## – CA Splits

- The number of buffers should be large enough to copy  $\frac{1}{2}$  of a CA at time
  - However, if the file does Mass Inserts or BROWSE operations, there is no way to segregate the buffers for one particular use



# NSR Performance

- **What is the best approach for files that are heavily or mainly browsed?**
  - If too many buffers are read, performance of other tasks may be affected
  - The key is to try and get a CISZ that generally accommodates the # of READNEXT commands issued
  - This approach can be used for LSR pool files too



# NSR Performance

- **Why would you want extra BUFNIs?**
  - **Two types of look asides occur in an NSR file**
    - **The 1<sup>st</sup> look aside is for the Index Set records that are in extra BUFNI buffers**
    - **The 2<sup>nd</sup> look aside is within the string buffers to see if the Sequence Set Index and/or the data CIs are present**
      - **No look aside possible in other string buffers**



# NSR Performance

- **Additional index buffers allows VSAM to load the Index Set records**
  - **User should allocate sufficient BUFNIs as there are Index Set CIs in the file**
  - **Consideration should be given to adding additional buffers if the file reflects CA splits**
    - **Data CA splits can cause index CA splits creating new index set records**



# NSR Performance

- **Determining the number of BUFNIs required entails computing how many Sequence Set records exist in the file**
  - **There is one Sequence Set record per data CA**
  - **This is a one to one relationship**



# NSR Performance

- **Compute:**
  - 1) **# CAs = (Data HURBA / (# CI/CA\*Data CISZ)** this represent the # of Sequence Set Index records in the file
  - 2) From LISTCAT get the total number of Index records in the file and determine the number of Index Set records in the file: (Total Number of Index Records -- # of CAs)
  - 3) Determine the # of BUFNIs = (Total # Of Index Set records + # of strings + CA split adjustment)
  - 4) CA Split adjustment is any figure from zero to “n”, where “n” is the # of additional Index set records created as a result of CA splits

**NOTE: IMPROPERLY CLOSED VSAM FILES COULD RESULT IN INCORRECT CALCULATIONS BECAUSE LISTCAT STATISTICS MAY NOT BE ACCURATE**



# NSR Performance

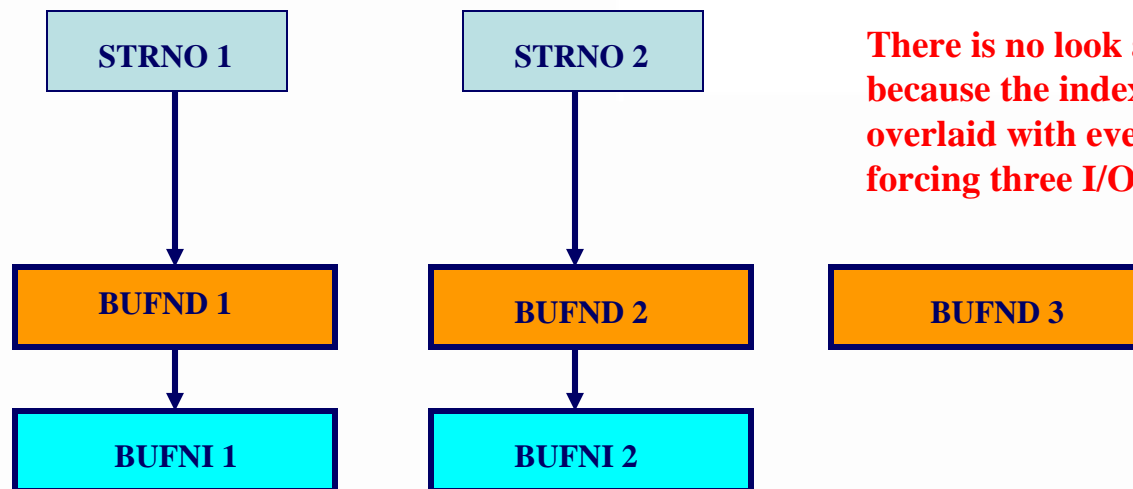
- **Example using a file LISTCAT information**
  - **Data CISZ**            **18K (18,432)**
  - **CI/CA**                **45**
  - **Bytes/CA**            **829,440**
  - **CA splits possible?**        **Yes**
  - **# of IX records**        **124**
  - **HURBA**                **97,873,920**
  - **# of IX levels**        **3**
  - **(97873920/(829440))=118 CAs or Sequence Set Recs**
  - **(124-118)=6 Index Set Recs**
  - **If STRNO=5, then (5+6+2)=13 BUFNI request for the file. The +2 is a buffer for future CA splits.**



# NSR Buffer Definition

- **Example # 1 – VSAM 2 Index Levels:**

- STRNO = 2      BUFND = 3      BUFNI = 2
- Requires three I/Os (2 index and 1 data)
- No opportunity for look aside



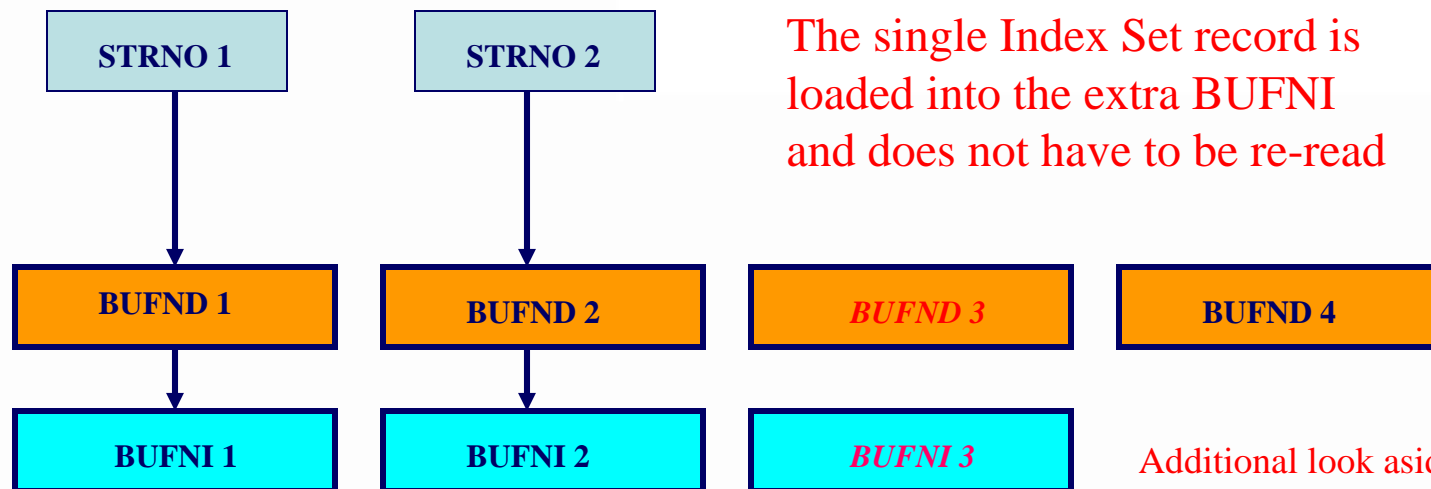
**There is no look aside possible because the index buffer gets overlaid with every request forcing three I/O operations**



# NSR Buffer Definition

- **Example # 2 – VSAM 2 Index Levels:**

- STRNO = 2      BUFND = 4      BUFNI = 3
- After 1<sup>st</sup> read, each request would require a maximum of two reads or a 33% I/O operations savings



# Recommendations

- **NSR files should be reviewed to see why they are not in LSR for better performance**
  - For example, Share Options 4 file
  - Command Level Browse restrictions
- **If the file is to be in NSR**
  - Ensure valid CISZ for files that are browsed
  - Ensure sufficient BUFNIs allocated to hold the entire Index Set indices in buffers
  - Ensure that excess strings are eliminated and the storage used to allocate correct file buffering
  - Do not over allocate BUFND unless the file is prone only to CA splits
- **If NSR must be used and files takes CA splits, be prepared to have CA splits affect CICS performance**



# Tuning CICS LSR Buffers



Copyright © 2007. CTREK Corporation. All Rights Reserved.

# Robin Hood and LSR

- **Robin Hood Philosophy:**
  - “ROB FROM THE RICH TO GIVE TO THE POOR”
- **Tuning LSR Is the Robin Hood Philosophy in Reverse**
  - Who are the “rich” files?
    - » Usually Most Active Files
    - » Usually Your “Bread and Butter” Files
  - Who are the “poor” files?
    - » Usually Least Active Files
    - » Usually Not Important Files
- **“ROB FROM THE POOR TO GIVE TO THE RICH!”**



# Pool Definition

- **Dynamic Pool Definition**
  - **Advantages**
    - **Allows for Quick Implementation and Installation**
    - **Reduces System Programmer Intervention**
      - **No Need to Compute CISZ vs. Buffers Required**
      - **No Need to Determine Maximum Key Length**
      - **No Need to Compute Number of Strings Required**
  - **Disadvantages**
    - **CISZ Contention Between Data and Index – Combined Pool**
    - **Allocation of Buffers Is Based on a Percentage Not Activity**
    - **String Allocation Based on % – Usually Over-allocated**
    - **Slow CICS Initialization (First File Opened)**
    - **Combined Data/Index Pools Can Hide Bad Data/Index Performers**



# Pool Definition

- **Static Definition**
  - **Disadvantages**
    - **Requires System Programmer Intervention to Determine**
      - **Buffers Sizes and Quantity Required**
      - **Maximum Key Length**
      - **Number of Strings Needed**
    - **Exposes System Programmer to Errors**
      - **Incorrect Buffer Size Selection – Buffer Fragmentation**
      - **Incorrect String Allocation**
      - **Incorrect Maximum Key Size Specification**
    - **Requires Planning – Not Everyone Likes to Do This!**
      - **Must Specify Required Buffers, Maximum Key Length and Number of Strings Required – Otherwise Pool Is Dynamically Created**



# Pool Definition

- **Static Definition**

- **Advantages**

- **Separate Pools for Data and Index Can Be Defined**
      - **No CISZ Contention Between Data and Index**
    - **Can Optimize Buffers that Have Higher Activity**
    - **Can Optimize String and Maximum Key Size Required**
    - **Faster Pool Initialization**



# Pool Definition

- **Recommendation**
  - **Define LSR Pools Explicitly**
  - **Determine Individual File Requirements**
    - Data and Index (If Applicable) CISZ required
    - Maximum Length Key
    - Strings
  - **Get “Big Picture” of Requirements**
    - CICS Performance Tool/Monitor
    - CICS Statistics (EOD)
    - Dynamic Definition – One Time to Get the “Big Picture”



# LSR Pool Measurement

- **LSR Pool Effectiveness Is Based on Look-Aside Hit Ratios**
  - **Generally Accepted Hit Ratios Are:**
    - Data – 80%
    - Index – 95%
    - Combined – 93%
- **Buffer Tuning Should Concentrate on Improving the Index Hit Ratio First**
  - **Generally, Index I/O Is Higher Than the Data**
  - **Virtual and Real Storage Investment to Improve Index Hit Ratio Is Less Due to Smaller CISZ Associated with the Index Component**



# LSR Pool Buffer Allocation

- **Generally, LSR Tuning Entails Adding Buffers to a Pool (Data/Index)**
- **Buffer Tuning Is Usually a “Trial and Error” Task**
  - **Review Buffers Requiring Attention and Add More**
  - **Re-Start LSR Pool and Measure Results**



# LSR Pool Buffer Allocation

- **Two Schools of Thought Regarding Buffer Allocation**
  - **School 1 – Allocate Buffers to All 11 Possible Buffer Combinations**
    - **Disadvantage – Wasted VS**
  - **School 2 – Only Allocate Buffers That Are Going to Be Used**
    - **Disadvantage – Possible Fragmentation Exposure**
    - **File May Not Open If Larger CISZ Buffer Is Not Available**



# LSR Pool Buffer Allocation

- **School 1 Example:**
  - No Files That Use a 12K, 16K, 24K or 28K LSR Buffers
  - Cover All Bases – So Assign a Minimum of 20 Buffers to Each Unused Buffer for Possible Future Use
  - If No File Is Ever Defined, Loss Is
    - 12K X 20 = 240K
    - 16K X 20 = 320K
    - 24K X 20 = 480K
    - 28K X 20 = 560K
  - Total Unused Storage = 1.6MB



# LSR Pool Buffer Allocation

- **School 2 Example:**
  - **Do Not Allocate Storage for Files That Do Not Use a 12K, 16K, 24K or 28K LSR Buffers**
  - **Instead, Use the Storage for Used Buffers**
    - **Take the 12K and 16K Buffer Storage (560K) and Allocate 28 20K Buffers**
    - **Take the 24K and 28K Buffer Storage (1040K) and Allocate 32 32K Buffers**
  - **These Buffers Will Be Used Every Day Providing Better Look-Aside for the Used Buffers**
  - **If in the Future a File That Needs the Un-Allocated Buffers, Would Use the Spare Buffers Assigned to the Larger Buffer**



# LSR Pool Data Buffers

- **Data Buffer Tuning Is Highly Dependent on Access Patterns**
  - **Good Look-Aside Hit Ratios Usually Requires a Substantial Storage Investment (80%+)**
  - **The Major Cause Is That the Data Component Is Usually Very Large (vs. Index Component)**
  - **Good Hit Ratios Usually Result in Files with:**
    - **Sequential Activity**
    - **Read for Update/Rewrite/Delete**
    - **Concentrated Read Activity**



# LSR Pool Data Buffers

- **Data Buffer Tuning Is Highly Dependent on Access Patterns**
  - **Bad Hit Ratios Usually Result in Files with:**
    - Disperse Read Activity (Very Large Files)
    - Share Options 4
- **Recommendation**
  - **Tune Buffer Pools and CI Sizes Individually**
    - **Set Realistic Objectives, for Example:**
      - Data – 80%
      - Index – 95%
      - Combined – 93%
  - **Define a Minimum of Three 32K Catch-All Buffers**



# LSR Pool Index Buffers

- **Tune Index Buffers First**
  - **Index Component Usually Has More Activity**
  - **Index Component Has Less CIs Than Data Component**
    - **Less LSR Buffers Required to Cover a Broader Base**
  - **Index CISZ Are Also Less Varied**
    - **Sizes Usually in the .5K to 4.0K Range**
  - **Aim to Get at Least 95%+**
  - **Define at Least Three 32K Buffers**



# LSR Pool Recommendation

- **Recommendation**
  - **Can Allocate Up to 32K Buffers Per LSR Buffer**
    - Hashing Search Used – No CPU Penalty
    - Don't Be Meek
  - **Probably Require a CICS Tuning Tool/Monitor**
  - **Review Periodically**
    - Tuning Is a Continuous Process
    - Control and/or Be Aware of Application Maintenance Procedures



# Overlooked LSR Tuning Areas

- **Buffer Fragmentation**
  - **Only Eleven Valid CISZ for LSR Buffers (K)**
    - 0.5   1.0   2.0   4.0   8.0   12.0
    - 16.0   20.0   24.0   28.0   32.0
  - **Therefore, a 2.5K Byte CISZ Would Use a 4K LSR Buffer**
  - **If a 4K Buffer Was Not Available, Then the Next Largest Available Buffer Is Used**
  - **Some Fragmentation May Be Desired for Certain CISZ (18.0K)**



# Overlooked LSR Tuning Areas

- **Buffer Fragmentation**
  - **Avoid Unnecessary Fragmentation (e.g., a 6K CISZ Using a 12K Buffer)**
  - **Certain Default Index CISZ Should Be Forced to an LSR CISZ (e.g., 1536 to 2048 or 2560 to 4096)**
  - **Virtual Fragmentation Results in Real Storage Fragmentation**



# Overlooked LSR Tuning Areas

- **LSR Buffer vs. File CISZ Reconciliation**
  - **Best Alternative to Reducing Fragmentation**
  - **Determine File CI Sizes Required and Assign LSR Pool Buffers to Match**
    - **Number and Size of Buffers**
    - **Number of Strings (Overall)**
  - **Set CISZ Standards (If possible) for LSR Pool Files**



# Overlooked LSR Tuning Areas

- **Allocated Storage vs. Return on Investment**
  - **Excessive Virtual and Real Storage to a Particular Pool May Not Yield Desired Return**
  - **Particularly True for Data Pools**
  - **Buffer Increases Should Result in at Least a 5% Increase in Hit Ratio**
  - **If ROI Is Insufficient, Then Re-Allocate Storage to Another Pool or Buffer**
  - **Fix Buffers That Have High Activity Even If % of Hit Ratio Is Poor for a Low Activity Buffer**



# Overlooked LSR Tuning Areas

- **Page Boundary Buffer Allocation**
  - VSAM Requests Buffers on a Page Boundary and in Page (4K) Increments
  - Fragmentation That Occurs from Buffer Allocation Should Be Avoided – Loss of Virtual Storage
  - Allocate the Following Buffers in the Following Multiples:
    - 0.5K Multiple of 8 (0.5K Times 8 = 4K)
    - 1.0K Multiple of 4 (1.0K Times 4 = 4K)
    - 2.0K Multiple of 2 (2.0K Times 2 = 4K)



# Overlooked LSR Tuning Areas

- **Buffer Pool Monopolization**
  - **Theory Behind LSR Is to Share Resources When Needed**
    - **So What Can Be Bad If the Principal Files (Most Active) Control a High Percentage of the Buffers?**
    - **Even at the Expense of Low Activity Files**
  - **How Do You Determine If a File Is Monopolizing the Buffer Pool?**
    - **I/O Activity**
    - **Buffer Hit Ratio**
    - **Number of Buffers Held (By CISZ)**



# Overlooked LSR Tuning Areas

- **Buffer Pool Monopolization**
  - **Need a CICS Tuning/Monitor to Determine the Number of Buffers Being Held by a File**
  - **Important If Principal Files Are Not Providing a Good Response Time**
  - **A Possible Solution to Buffer Monopolization Is to Allocate More Buffers to a Particular Size Even Though the Look-Aside Hit Ratio Is Meeting Objective**



# Overlooked LSR Tuning Areas

- **Maximum Key Size**
  - **Maximum Key Size Is Important as All VSAM Control Blocks Are Shared and Must Accommodate the Largest File Key of the Shared Pool**
  - **If the Maximum Key Size Allocated to the Pool Is too Small, Files with Larger Keys Will Not Open**
  - **Many Installations Force the LSR Pool Key Size to 255 Bytes**
  - **Although Using This Maximum Can Waste Storage, the Actual Amount Depends on the Number of Strings Allocated Times the Excess Key Size**
  - **Decision is Installation Dependent**



# Overlooked LSR Tuning Areas

- **Number of Strings Allocated**
  - **Probably Only Tuned When Wait on Strings Conditions Occur**
  - **Many LSR Buffer Pools are Over-Allocated**
  - **The Objective Should Be to Have Sufficient Strings to Handle Peak Periods Without Waiting for Strings**
  - **Try to Allocate So That the High Used String Number Is Around 40 to 60% of the Total Strings Allocated to the Pool**



# Overlooked LSR Tuning Areas

- **Number Of Defined LSR Pools**
  - **Two Schools of Thought**
    - **School 1 – Use as Many Pools as Possible So That Files Can Be Segregated to Reduce Contention and/or Interference**
    - **School 2 – Use as Few as Possible Pools So That Resources Can Be Used More Efficiently**
  - **Considerations**
    - **Are the Pools Allocated with a “Fudge Factor”?**
    - **Which Files Are More Important So That Resources Should Be Allocated to Them?**



# Overlooked LSR Tuning Areas

- **There Are 15 LSR Pools Available in LSR**
  - **Made Sense in the Beginning Because Buffer Search Algorithm Was Sequential**
  - **Larger Pools Increased CPU Time to Search**
  - **Search Algorithm Changed – Hashing Technique**
    - **Search Time is Consistent No Matter Number of Buffers**
- **Theory Behind LSR Is to Share Resources When Needed**
  - **So What Can Be Bad If the Principal Files (Most Active) Control a High Percentage of the Buffers?**
  - **Even at the Expense of Low Activity Files**
    - **Robin Hood Philosophy in Reverse**
- **Use a Separate Pool to Favor a File**



# LSR Pool Candidates

- **LSR Provides the Best Look-Aside Algorithm Within CICS/TS**
- **Generally, Files (High, Intermediate and Low Activity) Should Be Assigned to LSR Except:**
  - **Share Options 4 Files**
  - **Files That Do Not Follow Command Level Guidelines**
    - **Start Browse, Read Next .....Read for Update (Non-RLS)**
  - **High CA Split Activity Files (Tune Independently)**
- **LSR Is the Gate to New File Features Within CICS/TS**



# Closing

- **LSR Is Preferred Over NSR Buffering**
  - Superior Look-Aside Hit Ratio
- **Tuning LSR Involves:**
  - Ensuring Proper Number of Buffers Defined
    - Achieve Installation Look-Aside Hit Ratio Goals
  - Eliminating Fragmentation
  - Static Definition of the Pool(s)
- **Continuous Review – Especially When Major Application Changes Occur**
  - VSAM Tuning



# Transaction Controls and Tuning For CPU Resources



Copyright © 2007. CTREK Corporation. All Rights Reserved.

# Maximum Tasks--MXT

- **MXT – Maximum Tasks is the value that indicates the maximum number of non-system transactions allowed in CICS at any one time**
- **MXT is a global limit or control, that is, there is no discrimination established between transaction types. Each attached transaction counts toward the MXT total**
- **The maximum number of transactions allowed in the system is 999**



# Maximum Tasks--MXT

- **MXT can be used to control:**
  - Amount of Virtual Storage (VS) to be used -- by reducing or increasing this value, less/more transactions can be attached to use VS
  - Amount of Real Storage (RS) to be used – by reducing or increasing the value, less/more transactions can be attached to use RS
- **MXT is not a good control for specific type(s) of transactions**
  - MXT cannot be used to limit one or more types of transactions



# Maximum Tasks--MXT

- **Determining the MXT value for CICS is usually a trial and error process**
  - **Basis should be transaction rate per second**
  - **Must make allowances for long running transactions (e.g., browses, conversational etc.)**
  - **Setting MXT too low can cause delay conditions even though resources (CPU, Storage, and I/O) may be available (“Artificial Bottleneck”)**
  - **Setting MXT too high can result in over-commitment of resources (CPU, Storage, and I/O)**



# Maximum Tasks--MXT

- **Possible starting point:**

$$\text{MXT} = ((\text{Transactions/Second} * 1.50) + (\# \text{ of Long Running Transactions} * 1.25) + (\# \text{ of Conversational Transactions}) + 25)$$

**If Trans./Sec. < 1, use 6.00**

**If Conversational Trans. < 1, use 4**

**If MXT < 40, use 40**



# Maximum Tasks--MXT

- **Hidden Agenda**
  - **Transactions that must run below the line**
    - **DSALIM is a major factor if there are many programs that run below the line**
    - **DSALIM should be set to highest value possible in these cases**
    - **MXT may have to be set to a lower value or combined with TCLASS to guard against possible SOS conditions below the line**
    - **Each MXT slot has approximately 14K of VS pre-allocated (12K above and 2K below)**



# Maximum Tasks--MXT

- **Possible Solutions**
  - Increase the allocated **GETVIS** size
  - Increase current **EDSALIM** and **DSALIM** to maximum size possible
  - Use **TCLASS** to control high volume transactions below the line (Next topic)



# Maximum Tasks--MXT

- **Recommendations**
  - **Reaching MXT is not necessarily bad**
    - **Reaching MXT should never exceed 1% especially when there are sufficient resources available**
  - **Do not over-allocate MXT because serious problems can result such as SOS when unforeseen situations occur (e.g., deadlocks, string waits, slowdowns due to other system problems (e.g., paging) etc.)**



# Transaction Class-TCLASS

- **Transaction classes are available to provide additional granularity to executing transaction control**
- **There are usually four major reasons for using TCLASS:**
  - **Controlling resource “hogs”**
  - **Single threading to protect resources**
  - **Control number of transactions below the line**
  - **Avoid MRO “sympathy sickness”**
- **TCLASS is anonymous to the old CMAX specification except that TCLASSes are alphanumeric names versus numeric classes**



# Transaction Class-TCLASS

- **The major problem with using TCLASSes is that the use is inherited, that is, a transaction was assigned to a particular TCLASS many years ago when the system was short on resources and have never been updated with the new processor acquisitions**
- **As a result, transactions may be queuing even though resources exist—creating an artificial bottleneck**



# Transaction Class-TCLASS

- **Using TCLASS adds CPU overhead to the system**
  - Unnecessary controls add CPU usage
- **Lost resources are not recoverable**
  - Unnecessary use adds overhead
  - Artificial bottleneck
- **Use TCLASS only for transactions you need to control**



# Interval for Runaway--ICVR

- **The ICVR is a SIT supplied parameter that indicates the amount of time a transaction is allowed to execute before control (via an EXEC CICS command) has to be returned to the CICS dispatcher**
- **Not all EXEC CICS commands return control to the CICS dispatcher, e.g.,**
  - **GETMAIN**
  - **ASKTIME**



# Interval for Runaway--ICVR

- **The major problem with the ICVR is that it is usually inherited, that is, it was the value on the original SIT when CICS was converted to the new release**
- **The ICVR must take into consideration several factors:**
  - **Increased CPU speeds**
  - **Transaction rate/second**
  - **MXT specification**



# Interval for Runaway--ICVR

- **New processor speeds are well over 400 MIPS in range**
- **So, an ICVR=1000 (one second) would allow a user task to execute for at least 400M instructions before CICS could take action to cancel with an AICAabend**
- **Many installations use the IBM default for this parameter of ICVR=5000 (5 seconds)**



# Interval for Runaway--ICVR

- **Transaction rate per second is an important metric when determining the correct ICVR and MXT**
- **For example, suppose MXT=50 is set and the current transaction rate/second is 30**
  - **If ICVR=5000 is set, then 150 transactions would have queued up (5 seconds \* 30 transactions/second) before CICS would have cancelled a looping task with an AICA abend**
  - **If ICVR=1000, then only 30 transactions would have been queued**



# Interval for Runaway--ICVR

- **Recommendation – set ICVR to less than one second in today’s modern processors**
  - **If a transaction cancels with an AICA abend, consider assigning a separate value for this transaction on the transaction RDO definition**
    - **RDO parameter RUNAWAY defaults to “SYSTEM”, that is, use the SIT ICVR value**
    - **However, if you code a value (500-2700000) instead of using “SYSTEM”, then CICS will use the assigned value**



# 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



# Transaction Priority

- **Transaction priority is a means of providing dispatching preference for a particular transaction**
  - Priority can vary from 1 to 255
    - Priority is comprised of the sum of 3 elements to a maximum of 255 (truncated if greater):
      - Transaction priority (1-255)
      - Operator priority (1-255)
      - Terminal priority (1-255)
    - CICS dispatching is event driven (not interrupt) so the priority only affects the transaction's position on the ready queue
- **Priorities do not affect the sequence of servicing terminal requests**



# Transaction Priority

- **Use priority settings sparingly and should be set aside by classes, e.g.,**
  - **System Tasks → high 250 to 255**
  - **Normal Tasks → 1**
  - **Certain Important Tasks → 5**
- **If you are going to use PRTYAGE, do not use wide ranges for tasks**
  - **PRTYAGE only increments priority by one at the end of the PRTYAGE period**



# Transaction Priority

- **Consider high priority for:**
  - Using intrapartition transient data with logical recovery
  - Updating frequently used records
  - Automatic logging
  - Tasks needing fast application response time, for example, data entry.
  - Hold enqueues to resources
- **Consider lower priority for:**
  - Have long browsing activity
  - Are process-intensive with minimal I/O activity
  - Do not require terminal interaction, for example:
    - Auto-initiate tasks (except when you use transient data intrapartition queues that have a destination of terminal defined and a trigger level that is greater than zero).
    - Batch update controlling tasks.



# Transaction Priority

- **Recommendation**

- In general, only use transaction priority for system transactions
- If used, base priority on transaction and person, not terminal
- If used, do not use wide ranges of priorities when using PRTYAGE



# PRTYAGE

- **PRTYAGE is a SIT parameter that can be used to increase the priority of a non-dispatched transaction**
  - The PRTYAGE parameter is given in milliseconds and represents the amount of time that if a transaction is not dispatched, then its priority will be raised by one
  - Thus, if PRTYAGE=1000, then a non-dispatched transaction at the end of this period (1 second) would have its priority increased by one
  - Use of the PRTYAGE parameter changes the format of the dispatching priority value from a one byte figure ranging from 1-255 to an eight byte clock value
    - Tasks are then placed on the ready queue in a time sequence that includes time of arrival, dispatching priority and any penalties
    - If the PRTYAGE period expires, any non-dispatched task in this period has its eight byte priority increased by one
    - There is no transaction discrimination, that is, PRTYAGE applies to every task in the system
  - PRTYAGE also affects any task delays if system is under stress



# PRTYAGE

- **The question that has to be answered is:**
  - **Why am I using transaction priorities?**
    - **If the answer is to favor important transactions, then why should I want to dispatch a lower priority transaction over important transactions?**
- **The system has to be running at a very high CPU utilization for a task not to get dispatched**
- **So, the question is:**
  - **Why should I want to use precious CPU cycles to worry about the dispatching priority of low priority tasks?**
  - **Why not use these cycles to dispatch tasks instead of wasting them on accounting routines?**
- **If you want to use PRTYAGE, take the default of 32768 or 32.768 seconds**



# PRTYAGE

- **Recommendations:**
  - **Set PRTYAGE=0 to set off the algorithm and save CPU cycles**
  - **If you want to use PRTYAGE, then take the default of 32768 or 32.768 seconds**
    - **In this manner you can take the other delays (such as SOS delays), while not creating a significant overhead**



# Recommendations

- **Ensure that MXT is set correctly**
  - Reaching MXT with available resources is an “artificial bottleneck”
- **Do not use Transaction Classes unless there is a resource shortage and/or you want to single thread transactions (“artificial bottleneck”)**
- **Ensure that you have a built in safety valve against SOS conditions (stall) by using DTIMOUT and SPURGE**
- **Priorities should be used for critical performance and debugging transactions (high) and should be used to “control resource hogs” (low)**



# Tuning for CPU Resources

- **TUNING FOR CPU CYCLES IS NOT AN EASY TASK BECAUSE CICS IS A DISPERSE SYSTEM. A TRANSACTION HAS TO ACCESS MANY CONTROL BLOCKS, TABLES AND PROGRAMS TO EXECUTE**
- **SOME MAJOR CONTRIBUTORS TO CPU TIME**
  - I/O
    - EACH VSAM I/O ELIMINATED REPRESENTS BETWEEN 5K TO 8K SAVINGS IN INSTRUCTIONS (NORMALLY)
  - POOR APPLICATION CODE
  - INCORRECT PARAMETER SETTINGS
  - EXCEPTIONAL CONDITIONS (E.G., MXT, SOS, STRING WAITS etc.)
  - USE OF TRACE AND/OR MONITORING
- **CPU USE BY TRANSACTION IS A DETERMINING FACTOR IN HOW MANY TRANSACTIONS PER SECOND CAN BE SERVICED BY THE CICS SYSTEM**



# Tuning for CPU Resources

- **TUNING CICS FOR CPU CYCLES IS IMPORTANT BECAUSE DISPATCHING QUEUES BEGIN TO FORM WHEN THE CICS QR KTCB CPU UTILIZATION REACHES 60%**
  - **TRANSACTIONS THAT ARE READY TO EXECUTE WILL BEGIN TO FORM QUEUES AS THE CICS CPU CONSUMPTION INCREASES PAST 60%**
  - **MOST CICS CPU CONSUMPTION OCCURS ON THE CICS KTCB CALLED THE “QR” (90%+)**
  - **CICS APPEARS TO BE AN OPERATING SYSTEM BUT DOES NOT FUNCTION LIKE ONE**
    - **INTERRUPT VERSUS EVENT DRIVEN**
  - **CICS SYSTEMS THAT ARE RUNNING ABOVE 60% CPU UTILIZATION OF ONE CPU SHOULD BE SPLIT INTO MRO SYSTEMS TO OBTAIN MORE CPU OVERLAP**
  - **ANOTHER ALTERNATIVE IS TO TUNE THE CICS SYSTEM TO REDUCE THE OVERALL CPU REQUIREMENTS BY TASK**



# SIT PARAMETERS

## ICV

- **ICV IS USED TO IDENTIFY HOW LONG CICS SHOULD WAIT BEFORE BEING AWAKENED BY THE OPERATING SYSTEM**
  - ANY PENDING EVENT THAT IS COMPLETED WILL AWAKE CICS BEFORE THE ICV INTERVAL IS COMPLETE
  - “OLD” IDEA WAS TO WAKE CICS EVERY SO OFTEN TO PROTECT ITS WORKING SET AND REDUCE PAGING
  - CICS IS TOO LARGE AND DISPERSE FOR THIS FUNCTION TO WORK EFFICIENTLY PLUS AVAILABILITY OF LARGE REAL STORAGE HAVE MADE OBSOLETE THIS IDEA
    - IF THE PROBLEM IS A SHORTAGE OF REAL STORAGE THAT RESULTS IN PAGING, THEN TUNING FOR REAL STORAGE SHOULD BE PERFORMED BEFORE THIS PARAMETER IS LOWERED
  - OVERHEAD AFFECTS LOWER PRIORITY JOBS AND WASTES CPU TIME
  - CONSIDERATIONS ARE:  $MROBTCH > 1$ 
    - WAIT FOR MROBTCH IS SATISFIED WHEN THE # OF EVENTS SPECIFIED IN MROBTCH IS EQUAL, WHEN THE ICV TIME IS MET OR 3 SECONDS WHICHEVER OCCURS FIRST
  - GENERALLY, RECOMMENDED VALUES ARE 5000 ms. AND HIGHER



# SIT PARAMETERS

## ICVTSD

- **PROBABLY THE POOREST PARAMETER DEFINITION IN THE MANUAL**
- **THE ICVTSD INDICATES THE FREQUENCY WITH WHICH CICS TRIES TO DISPATCH THE TCP/ZCP FOR OUTPUT REQUESTS**
- **THIS VALUE DEFINES THE AMOUNT OF TIME TCP/ZCP MUST WAIT TO PROCESS:**
  - **NON-VTAM I/O REQUESTS WITH WAIT**
  - **NON-VTAM OUTPUT DEFERRED UNTIL TASK END**
  - **AUTOMATIC TRANSACTION INITIALIZATION REQUESTS**
  - **CICS REGION WITH A HIGH TRANSACTION ACTIVITY**
- **FOR VTAM ONLY REGIONS SET ICVTSD=0 WHICH FORCES THE TCP TCA (CSTP) TO BE CONTINUOUSLY DISPATCHABLE**



# SIT PARAMETERS

- **RELIABILITY FEATURES PROVIDE ADDED PROTECTION TO THE CICS ENVIRONMENT THAT MAY JUSTIFY THE ADDED OVERHEAD**
  - STORAGE PROTECTION (NEGLIGIBLE OVERHEAD) (STGPROT)
  - COMMAND PROTECTION (UP TO 1% OVERHEAD) (CMDPROT)
  - REENTRANT PROGRAM PROTECTION (NEGLIGIBLE OVERHEAD) (RENTPGM)
- **ALL THE PROTECTION FEATURES SHOULD BE TURNED ON IN A TEST ENVIRONMENT**
  - THIS PROVIDES A CERTAIN AMOUNT OF TRANSACTION TESTING PRIOR TO MOVING TO PRODUCTION ENVIRONMENT
- **PRODUCTION SYSTEMS SHOULD CONSIDER TURNING ALL OPTIONS ON**



# SIT PARAMETERS

- **TRACE IS A MAJOR CONTRIBUTOR TO CPU OVERHEAD IN AN EXECUTING SYSTEM AND CAN ACCOUNT TO AN ADDITIONAL 10-25% OVERHEAD**
  - IT IS NOT EASY TO TURN OFF THE TRACE COMPLETELY AS SOME PERFORMANCE MONITORS ARE BASED ON THE TRACE ENTRIES
  - ADDITIONAL OVERHEAD IS ADDED WHEN PROGRAMMERS PASS PROGRAMS INTO PRODUCTION CONTAINING TRACE ENTRIES
  - AVOID TRACE ID IN PRODUCTION SYSTEMS
- **BEST OPTION IS TO TURN OFF TRACE IN A PRODUCTION ENVIRONMENT**
  - IF NOT POSSIBLE, THEN CONTROL THE TYPE OF TRACE ENTRIES TO BE MONITORED
  - CONSIDER TURNING OFF IN “PURE” TORs, FORs AND/OR DORs
- **USE AUXILIARY TRACE ONLY FOR EXTREME PROBLEM DETERMINATION SITUATIONS OR PERFORMANCE PROBLEMS**
  - NEVER LEAVE AUXILIARY TRACE ON IN A PRODUCTION ENVIRONMENT WITHOUT A VALID REASON
- **TRACE TABLE SHOULD BE LARGE ENOUGH TO CAPTURE ONE SECOND OF DATA**



# SIT PARAMETERS

- **MONITORING ADDS CPU OVERHEAD TO THE RUNNING SYSTEM AND CAN ACCOUNT FOR A 3-11% OVERHEAD DEPENDING ON THE LEVEL OF MONITORING**
  - **THE MONITORING FEATURE MAY BE REQUIRED FOR PERFORMANCE MONITORS**
- **DMF CAN ADD 1.5% TO THE TRANSACTION CPU REQUIREMENTS**
- **TURN MONITORING OFF IF NOT REQUIRED**



# SIT PARAMETERS OTHER CONDITIONS

- **OTHER CONDITIONS THAT MAY AFFECT CPU UTILIZATION ARE:**
  - FEPI=YES → TURN OFF IF NOT USED
  - DO NOT TURN ON XRF UNLESS NEEDED – POSSIBLE 1-2% CPU OVERHEAD
  - DEVICE DEPENDENT SUFFIX (DDS) → DEFAULT, CAUSES UNNECESSARY CALLS TO THE LOADER DOMAIN
  - ELIMINATE UNNECESSARY DUMPS (e.g., ATNI, AP0001, SR0001)
  - ENSURE TASK/TERMINAL STORAGE CHECKS ARE SET OFF
  - PRIORITY AGEING REQUIRES PLANNING FOR EFFECTIVENESS
    - EXTRA CYCLES ARE QUESTIONABLE WHEN SYSTEM IS AT HIGH UTILIZATION
    - IF PRIORITY DIFFERENCES ARE HIGH, THEN THE NUMBER OF CYCLES REQUIRED FOR THE PRIORITY TO BE UPDATED SUFFICIENTLY FOR THE TASK TO BE DISPATCHED
  - OVER-ALLOCATED RAPOOL
  - ELIMINATE EXCESSIVE SECURITY CHECKING (MAY NOT BE DP DEPENDENT)
  - TAKING INTERVAL STATISTICS TOO OFTEN CAUSES ADDITIONAL OVERHEAD ON THE SYSTEM—TARGET FOR 6 HOUR INTERVALS (DEFAULT EVERY 3 HRS.)
  - SEC=YES CAN ADD UP TO 6% ADDITIONAL CPU PER TRANSACTION
  - MROLRM AND MROFSE CAN BE USED TO REDUCE CPU OVERHEAD IN THE DASD OWNING SYSTEM FOR REPETITIVE I/O OPERATIONS BY SAME TASK
- **THESE OPTIONS ALL ADD CPU OVERHEAD TO THE SYSTEM**



# TEMPORARY STORAGE

- **TUNING TS MAIN IS MAINLY ENSURING SUFFICIENT EDSALIM ALLOCATION AND A SUFFICIENTLY LARGE PARTITION SIZE**
- **MOST TUNING TO TS IS TO DFHTEMP (AUXILIARY STORAGE)**
  - **WRITES GREATER THAN CISZ ADD CPU OVERHEAD**
    - IF ((# OF WRITES GT CISZ / # OF AUX WRITES) \* 100) EXCEEDS 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
- **INCREASING # OF STRINGS TO RESOLVE STRING WAIT CONDITIONS MAY NOT BE THE APPROPRIATE SOLUTION**
  - **TUNE # OF BUFFERS FIRST AS BUFFERS ARE ONLY WRITTEN TO DFHTEMP WHEN A BUFFER IS REQUIRED**
  - **DELAYED BUFFER WRITES CAN INCREASE THE ACCESS TO TS QUEUES WITHOUT HAVING TO DO AN I/O—IMPROVED RESPONSE FOR AUX REQUESTS (NON-RECOVERABLE QUEUES)**
  - **THE LOOK-ASIDE HIT RATIO IS OPPOSITE THAN THE LSR HIT-RATIO → IN THIS CASE THE LOWER THE %, THE BETTER THE HIT RATIO**
- **TS FORMAT WRITES INDICATE THAT DFHTEMP HAD TO BE EXTENDED**
  - **FORMAT WRITES SHOULD NOT OCCUR**
  - **INCREASE THE PRIMARY ALLOCATION FOR DFHTEMP**



# TRANSIENT DATA

- **TUNING TRANSIENT DATA IS SIMILAR TO TUNING TS (DFHINTRA)**
  - TUNE BUFFERS FIRST INSTEAD OF STRINGS
  - ADDITIONAL BUFFERS CAN DELAY I/O OPERATION (NON-RECOVERABLE QUEUES)
- **FORMAT WRITES ON DFHINTRA SHOULD NOT OCCUR**
  - ENSURE VALID PRIMARY ALLOCATION
- **TUNE BUFFER ALLOCATION FOR EXTRAPARTITION DATA SETS**
  - TD IS USED IN MANY CASES TO PROCESS SAM FILES
  - OPERATING SYSTEM WAITS OCCUR WHEN REFRESHING THE BUFFERS
  - ADDITIONAL BUFFERS CAN REDUCE THE NUMBER OF WAITS PROCESSING THE FILE



# PROGRAM MANAGER

- **PROGRAM LOADS/COMPRESSION ADD ADDITIONAL CPU OVERHEAD**
  - ENSURE LARGEST POSSIBLE DSALIM/EDSALIM ALLOCATIONS
  - MAXIMUM POSSIBLE SIZE FOR DSALIM/EDSALIM
    - CICS WILL ALLOCATE ONLY WHAT IT NEEDS
  - THERE SHOULD BE ZERO PROGRAM COMPRESSIONS ABOVE THE LINE
    - MAKE EDSALIM AS LARGE AS POSSIBLE (400 TO 600 MB)
- **ELIMINATE UNNECESSARY LOAD WITH HOLD CONDITIONS WITHOUT AN INTERVENING RELEASE**
  - INCREASES THE OVERHEAD TO PROCESS THE SYSTEM LLE LIST



# OPERATING SYSTEM

- **ENSURE CICS HAS THE HIGHEST PRIORITY POSSIBLE WITHIN VSE**
  - CICS PRIORITY SHOULD BE JUST BELOW VSE VTAM AND PERFORMANCE/DEBUGGING PACKAGES
  - VERIFY THAT CICS HAS LITTLE OF NO PAGING DURING PEAK HOURS
    - MAX 5 TO 10 PAGE INS/SECOND
- **ENSURE CICS HAS SUFFICIENT VIRTUAL STORAGE AVAILABLE**
  - EXPAND DSALIM/EDSALIM VIA CEMT
  - ADD BUFFERS/STRINGS
- **ENSURE VALID LE OPTIONS**
  - USE 4080 BYTES FOR SECONDARY STORAGE ALLOCATIONS



# APPLICATION TUNING

- **THIS AREA IS USUALLY HARD TO IMPLEMENT BECAUSE OF THE DIFFERENT PRIORITIES SET**
  - Application Programming Objective Is to Get the Application into Production
- **BEST OPTION IS TO ADDRESS INSTALLATION OPTIONS**
  - **SOME COBOL CODING GUIDELINES**
    - NUMERIC FIELDS DEFINED A PACKED, SIGNED & ODD # OF DIGITS
    - USE CHARACTER FORMAT (X) FOR 1 BYTE CODES INSTEAD OF NUMERIC (9)
    - USE INDICES INSTEAD OF SUBSCRIPTS FOR TABLE SEARCHES
    - IF SUBSCRIPTS, DEFINE HALF-WORD BINARY
    - IN-LINE CODE
  - **CICS OPTIONS (IF YOU CAN INFLUENCE)**
    - AVOID BMS PAGING
    - AVOID STARTBR/READNEXT/ENDBR BY USING GENERIC READ
    - AVOID ALTERNATE INDICES—DO YOUR OWN CODING
    - USE TS MAIN AS MUCH AS POSSIBLE
  - **REMOVE ALL COMMAND LEVEL DEBUGGING FACILITIES (e.g., TRACE ID) FROM PRODUCTION PROGRAMS**
- **REMEMBER THAT YOUR APPLICATION USUALLY CONSISTS OF A FEW PROGRAMS THAT CALL MANY CICS SERVICE MODULE PROGRAMS**



# RECOMMENDATIONS

- **MAJOR FACTORS IN REDUCING CPU UTILIZATION ARE:**
  - **BETTER LOOK-ASIDE HIT RATIO (SAVE 5K TO 8K INSTRUCTIONS)**
    - NSR – BETTER BUFNI ALLOCATION
    - LSR – BETTER BUFFER ALLOCATIONS, SEGREGATING INDEX FROM DATA
  - **AVOID HITTING THRESHOLD LIMITS**
    - MXT
    - STRING/BUFFER WAITS
    - TCLASS
    - EXCLUSIVE CONTROL CONFLICT
  - **REDUCE OVERALL SYSTEM CLOCK INTERRUPT PROCESSING**
    - ICV
    - ICVTSD
    - ICVR
    - PRTYAGE
    - FEPI
  - **REDUCE I/O TO DFHTEMP AND DFHINTRA THROUGH MORE BUFFERS**
  - **TURN OFF TRACE AND MONITORING**



# Closing

- **Any Questions?**



Copyright © 2007. CTREK Corporation. All Rights Reserved.