SQL Server Performance Assessment and Optimization Techniques

Jeffry A. Schwartz

Windows Technology Symposium

December 6, 2004

Las Vegas, NV

jeffstx3@frontiernet.net

Emphasis of Presentation

- Interpretation and usage of informative performance counters
- Expand upon PerfMon explanations
- ALL graphs of actual customer data
- Insights acquired from analysis of many customer data sets
- Possible courses of action
- SQL Profiler usage considerations

Overview

- **SQL Server**
  - Measures many activities, but only certain ones can be traced to a **specific** database
  - None can be traced back to a specific query
  - Sometimes need additional tools such as SQL Profiler (SQL Server trace) to complete analyses
- **Must monitor other performance objects**
- **Presentation refers to SQL Server objects unless otherwise noted**

UNISYS
Imagine it. Done.

Overview

- **Many PerfMon explanations useless**
- **Example explanations**
  - SQL Compilations/sec is “Number of SQL compilations”
  - Table Lock Escalations/sec is “The number of times locks on a table were escalated”
  - Bulk Copy Rows/sec is “Number of rows bulk copied per second”

UNISYS
Imagine it. Done.

PerfMon Counter Hierarchy

- **Three-level hierarchy**
- **Objects at top level**
- **Counters**
  - Comprise bottom level
  - Always pertain to a particular object
- **Instance level added between object and counter levels when necessary**

Object Hierarchy Examples

- Processor object → Processor 0 instance → % Processor Time counter
- Memory object → Page writes/sec counter

UNISYS
Imagine it. Done.

SQL Server Objects

- One set per SQL Server instance
- Each set divided into 17 categories
- 4 memory-related
- 2 lock-related

UNISYS
Imagine it. Done.

SQL Server Objects

- **7 measure database backup, replication, and user settable categories**
  - Applicable to database backup and replication performance, as well as specifically defined and maintained user counters
- **Rest involve database transactions, log handling, and database access activities**

UNISYS
Imagine it. Done.

SQL Server Object List

SQL Server Objects	Category
SQL Server: Access Methods	Database access
SQL Server: Backup Device	Database backup
SQL Server: Buffer Manager	Memory management
SQL Server: Buffer Partition	Memory management
SQL Server: Cache Manager	Memory management
SQL Server: Databases	Transactions & log handling
SQL Server: General Statistics	User connections
SQL Server: Latches	Locking
SQL Server: Locks	Locking
SQL Server: Memory Manager	Memory management

SQL Server Object List

SQL Server Objects	Category
SQL Server: Replication Agents	Database replication
SQL Server: Replication Dist.	Database replication
SQL Server: Replication Logreader	Database replication
SQL Server: Replication Merge	Database replication
SQL Server: Replication Snapshot	Database replication
SQL Server: SQL Statistics	SQL command activities
SQL Server: User Settable	User defined

UNISYS
Imagine it. Done.

Buffer Manager and Buffer Partition Objects

- 21 counters
- 5 involve Address Windowing Extensions (AWE)
- AWE covered indirectly

UNISYS
Imagine it. Done.

Buffer Cache Hit Ratio

- Frequency with which database read requests are satisfied from database cache memory instead of disk
- Higher values result in lower disk usage
- Recommended value at least 90%
- Raw performance data can sometimes exceed 100%

UNISYS
Imagine it. Done.

Detecting Insufficient SQL Memory

- Compare Memory Manager object's *Target Server Memory (KB)* with *Total Server Memory (KB)* counters
- If Total less than Target, possibly insufficient memory
- If comparison too small or *Buffer Cache Hit Ratio* is too low, allocate more memory to SQL Server, if possible

UNISYS
Imagine it. Done.

SQL Server Only App and Single Instance

- **If SQL Server only application on system and there is only one instance, decisions may be simpler**
  - Reconfigure to use all of memory automatically, if not already doing so and **not** using AWE
  - Add more memory

AWE and SQL Server Using All Memory Automatically

- **Setting SQL Server to use all memory has often caused problems when AWE used on Windows 2000**
- **Can cause system to**
  - Exhaust Windows memory
  - Page heavily
- **AWE-related tables hard to identify**

UNISYS
Imagine it. Done.

AWE and SQL Server Using All Memory Automatically

- Scalability experts have stated that AWE-related tables consume more Windows memory as more AWE memory locations accessed (at least on Windows 2000)
- Experience has shown this to be true

UNISYS
Imagine it. Done.

AWE User Experiences #1

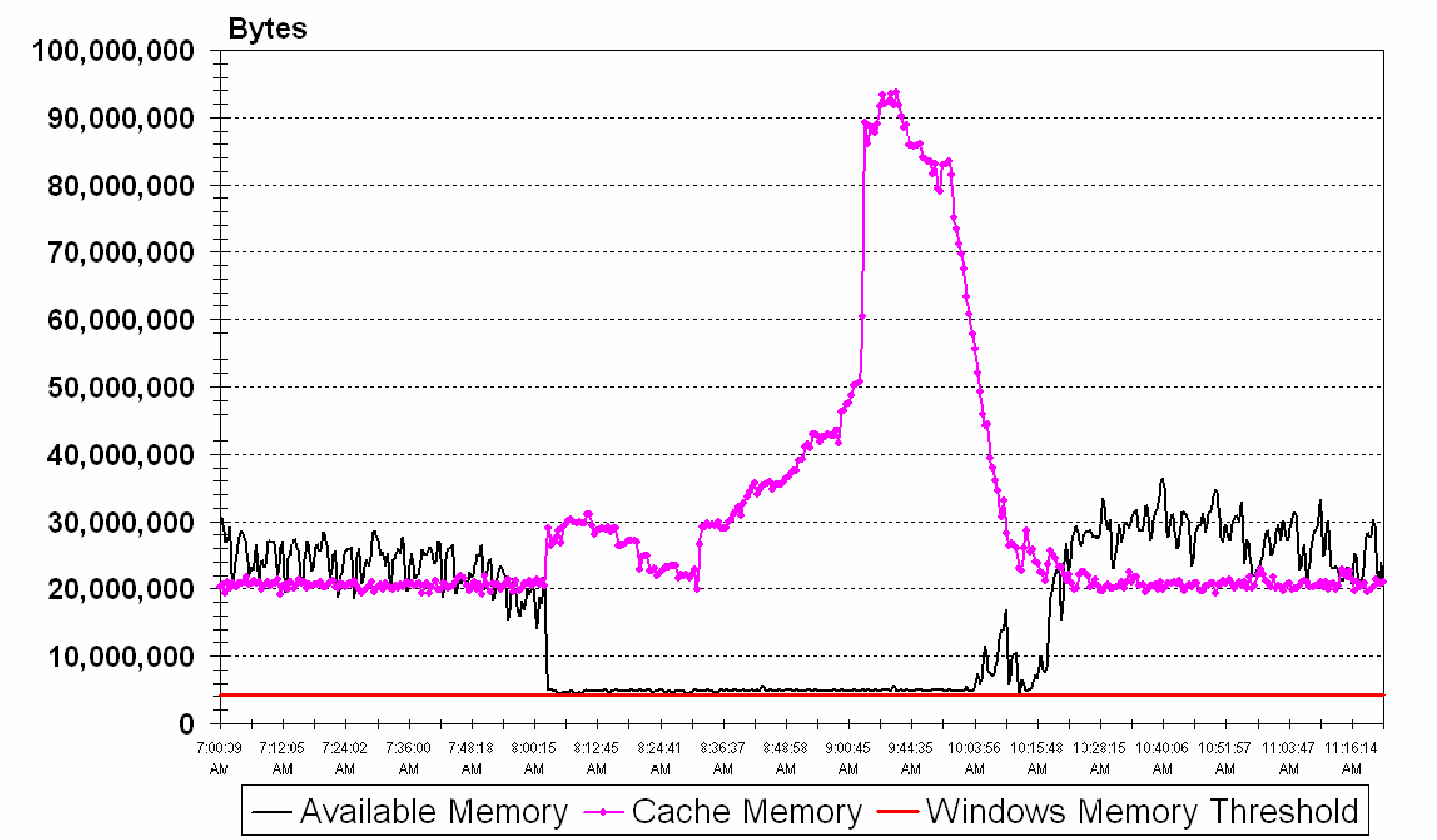
- **30 GB allocated to SQL Server on 32 GB system**
  - System paged heavily after user activity increased
  - Little or no available memory for Windows
  - Could not attribute Windows memory usage to a process
- **Reduced allocation to 28 GB**
  - Paging ceased
  - Both system and SQL Server ran fine
  - Buffer cache hit ratio hardly affected

AWE User Experiences #2

- **All but 65 MB allocated to SQL Server on 8 GB system**
  - System ran fine for several weeks, but available memory decreased very slowly
- **Full text index creation executed**
  - System cache increased
  - Index creation programs required own non-SQL memory
- **System began to page heavily**
  - Little or no available memory for Windows
  - System basically stopped functioning when available memory dropped below 4 MB
- **SQL memory allocation reduced to insure 678 MB available – problems ceased permanently**

UNISYS
Imagine it. Done.

Available & Cache Memory



SQL One of Many Apps

- **Decisions much more complex if SQL Server NOT only major application on computer or multiple instances exist**
- **Classic system versus database conflict**
  - Allocating too much memory to SQL Server can harm other applications, SQL Server instances, or Windows, unless sufficient memory can be added
- **Need to match instance memory with business requirements**

Increasing SQL Memory

- **Make very gradual changes**
- **Monitor system Memory object counters before and after any changes**
  - Page writes/sec
  - Available Bytes
    - Insure Windows 4 MB available memory limit **impossible to reach**, regardless of application activities

Free Pages Counter

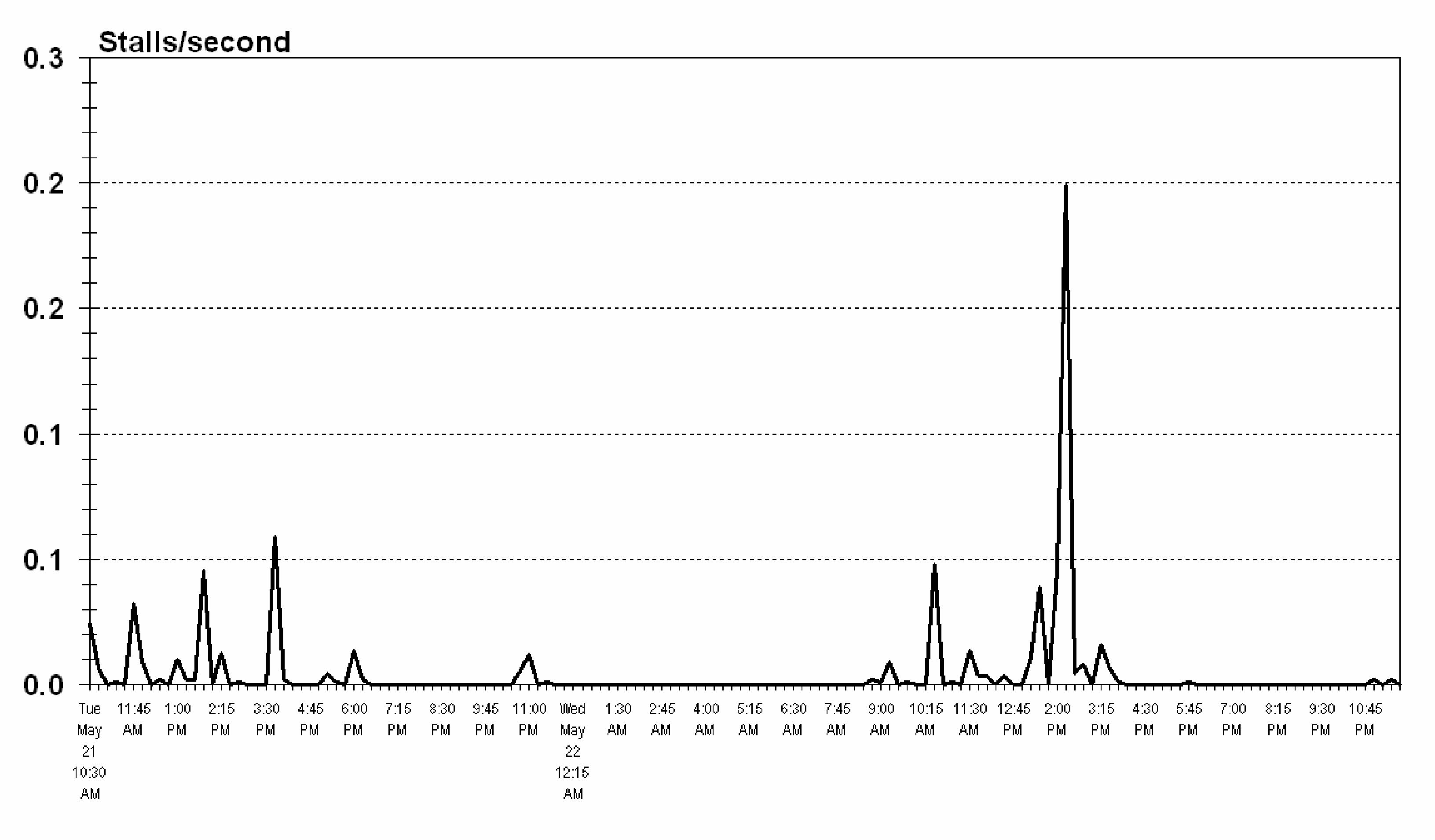
- Number of memory buffers available to receive database pages read from disk
- Indicator of insufficient SQL Server memory
- Values consistently close to zero indicate SQL Server memory shortage
- Closely associated with *Free list stalls/sec*

FreeList Stalls Counter

- Frequency with which requests for available database pages are suspended because no buffers are available
- Free list stall rates of 3 or 4 per second indicate too little SQL memory available

UNISYS
Imagine it. Done.

FreeList Stalls Graph



UNISYS
Imagine it. Done.

Stolen Pages Counter

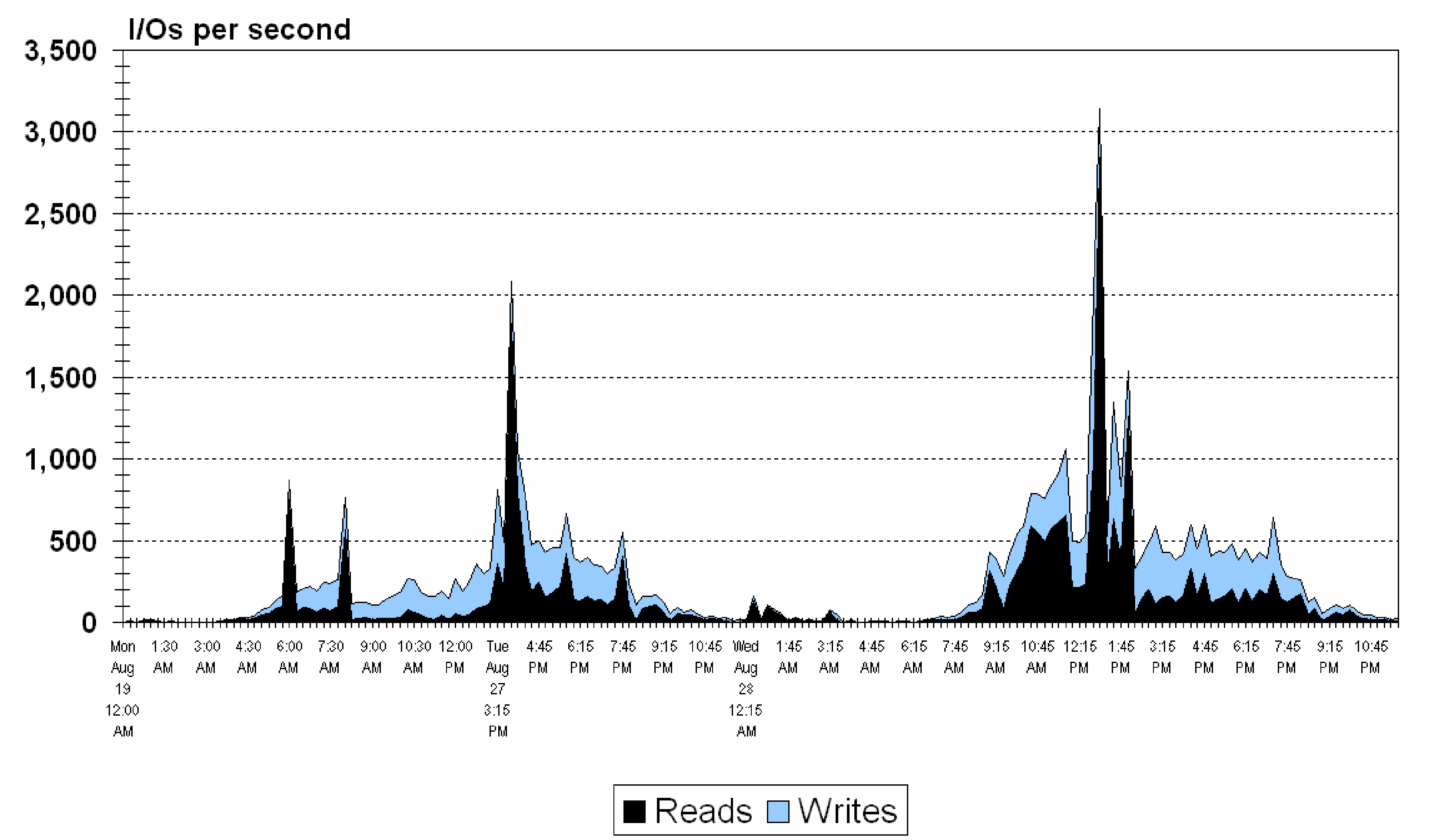
- Pages "stolen" when Windows requires memory for another application
- Useful indicator of overall system memory shortage
- Short periods may be normal
- Example: system backup begins after large database batch run completes

Database I/O Counters

- ***Page Reads/sec*** **and** ***Page Writes/sec*** **counters**
- **Measures physical I/Os, not logical I/Os**
- **May indicate**
  - Insufficient database memory
  - Applications improperly accessing database
  - Improper database table implementation

UNISYS
Imagine it. Done.

I/O Activity Graph



UNISYS
Imagine it. Done.

Page Lookups/sec Counter

- **Measures number of times database attempted to find a page in buffer pool**
- **Logical read**
- **Useful for corroborating and further quantifying buffer cache hit ratio**
- **Compare *Page Reads/sec* with *Page lookups/sec***

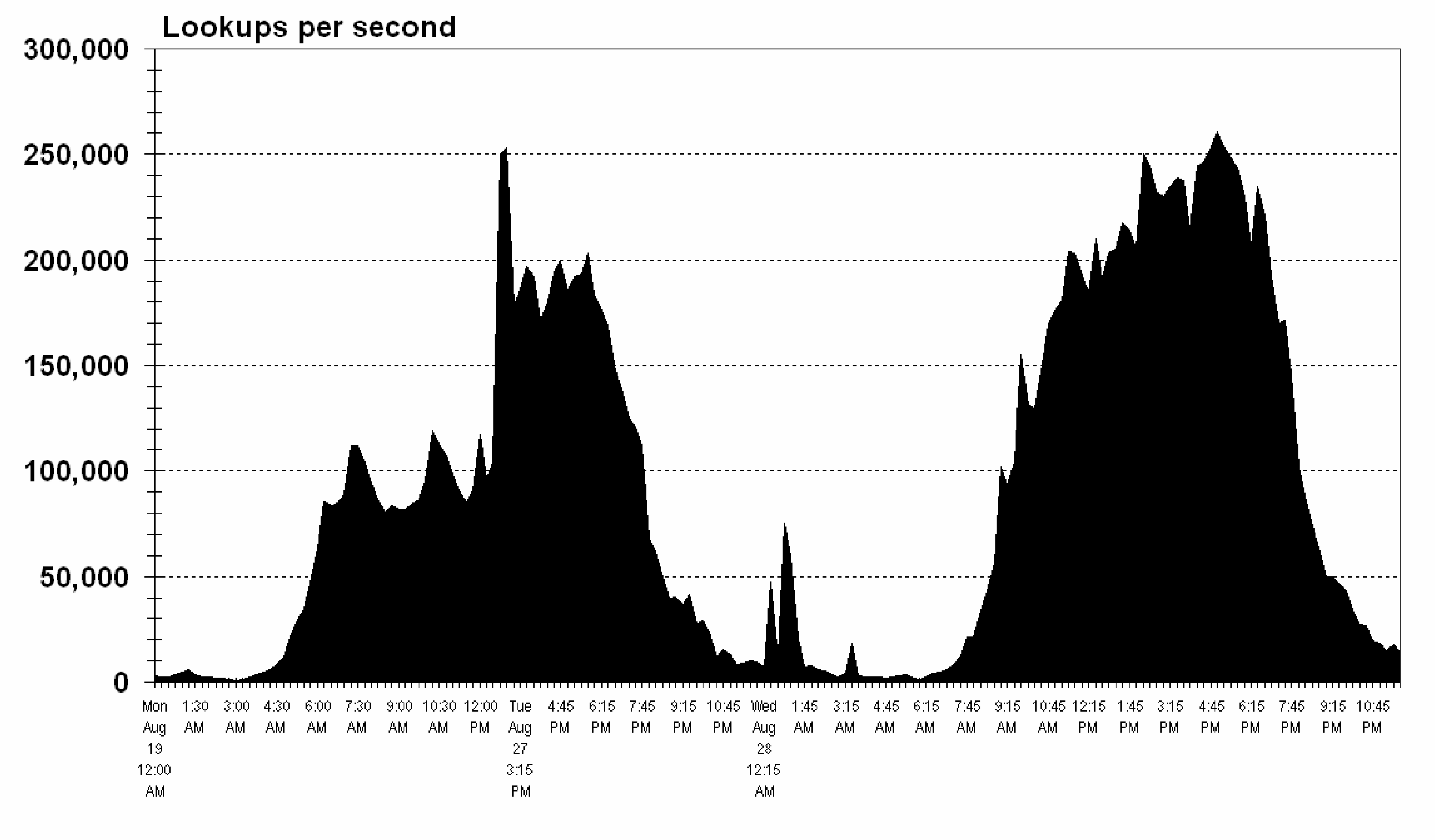
UNISYS
Imagine it. Done.

Buffer Cache Hit Ratio - Revisited

- Can perform computation when more precision necessary, e.g., 30 of 32 GB allocated to SQL
- 1 – (Page reads/sec / Page lookups/sec)

UNISYS
Imagine it. Done.

Page Lookups Graph



Memory Manager Object

- **Counters can be used to develop SQL Server memory composition graph**
  - *Connection Memory (KB)*
  - *Granted Workspace Memory (KB)*
  - *Lock Memory (KB)*
  - *Optimizer Memory (KB)*
  - *SQL Cache Memory (KB)*
- **Monitor lock blocks**

UNISYS
Imagine it. Done.

Access Methods Object

- **Most helpful counters**
  - *Forwarded Records/sec*
  - *Full Scans/sec*
  - *Index Searches/sec*
  - *Range Scans/sec*
  - *Table Lock Escalations/sec*

UNISYS
Imagine it. Done.

Forwarded Records

- **Only occur in tables without clustered indices, i.e., heaps**
- **Occur when row/record moved from one database page to another because changed record cannot fit back in original page**
  - Image data, i.e., bitmap data
  - Variable-length string data
- **Most frequently occur in Tempdb**

Forwarded Records

- **Creates de facto physical linear search, which can cause *long* record access times and high page read rates**
- **Adding clustered index is simplest way to eliminate problem**
  - Use as few data columns as possible
- **Otherwise, create records that are large enough to accommodate changes**

UNISYS
Imagine it. Done.

Detecting Forwarded Records

- **Two ways to determine total count of forwarded records in a table**
- **Enable trace flag 2509 and execute DBCC CHECKTABLE command as shown below**
  - DBCC TRACEON (2509)
  - GO
  - DBCC CHECKTABLE (<table name>)

OR

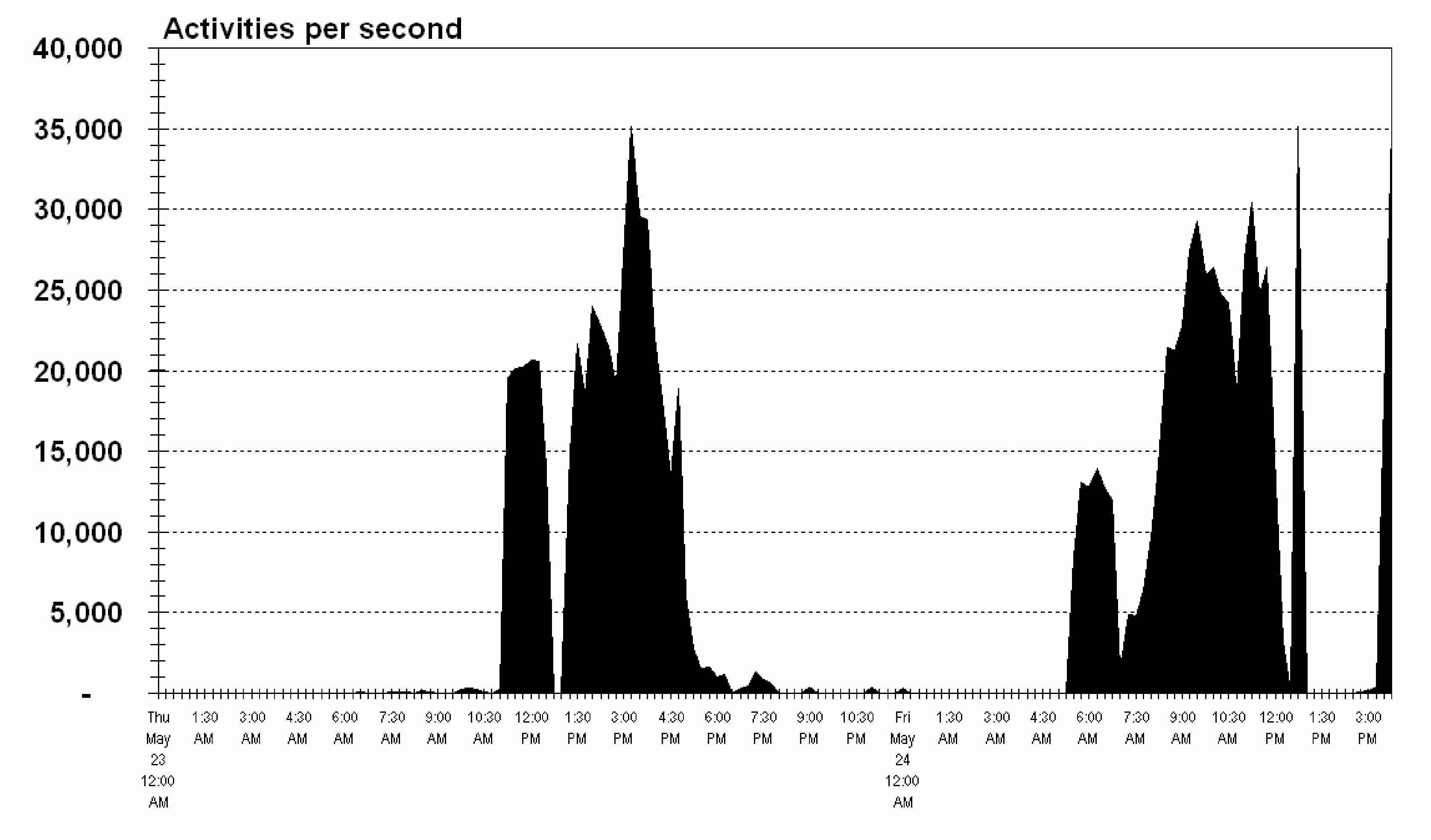
- **Execute DBCC SHOWCONTIG using TABLERESULTS option as shown below**
  - DBCC SHOWCONTIG (<table name>) WITH TABLERESULTS

Forwarded Records/sec

- Measures # of records fetched via forwarded record pointers
- Since forward record "chains" are prevented by SQL Server, counter refers to actual record count, not number of pointer "chases"

UNISYS
Imagine it. Done.

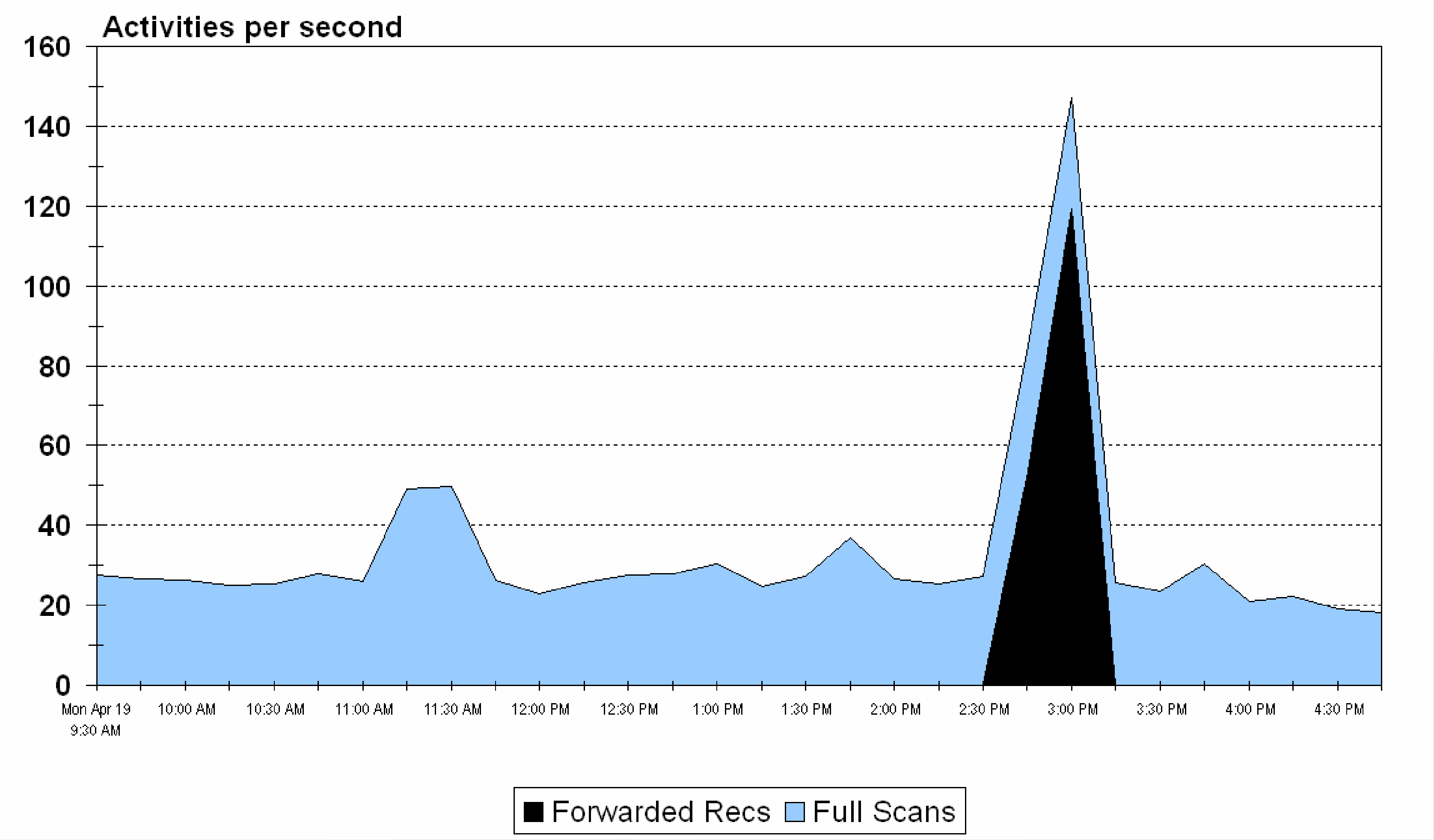
Forwarded Records Graph



Full Scans

- Unrestricted linear searches through table or index
- Example SQL statement
  - SELECT * FROM TABLETHATISAHEAP
- SQL Query **Estimated Execution Plan** can identify them ahead of time
- SQL Query **Actual Execution Plan** and SQL Profiler (trace) can identify them when they occur
- Trace records contain **logical** reads and writes

Access Method Graph

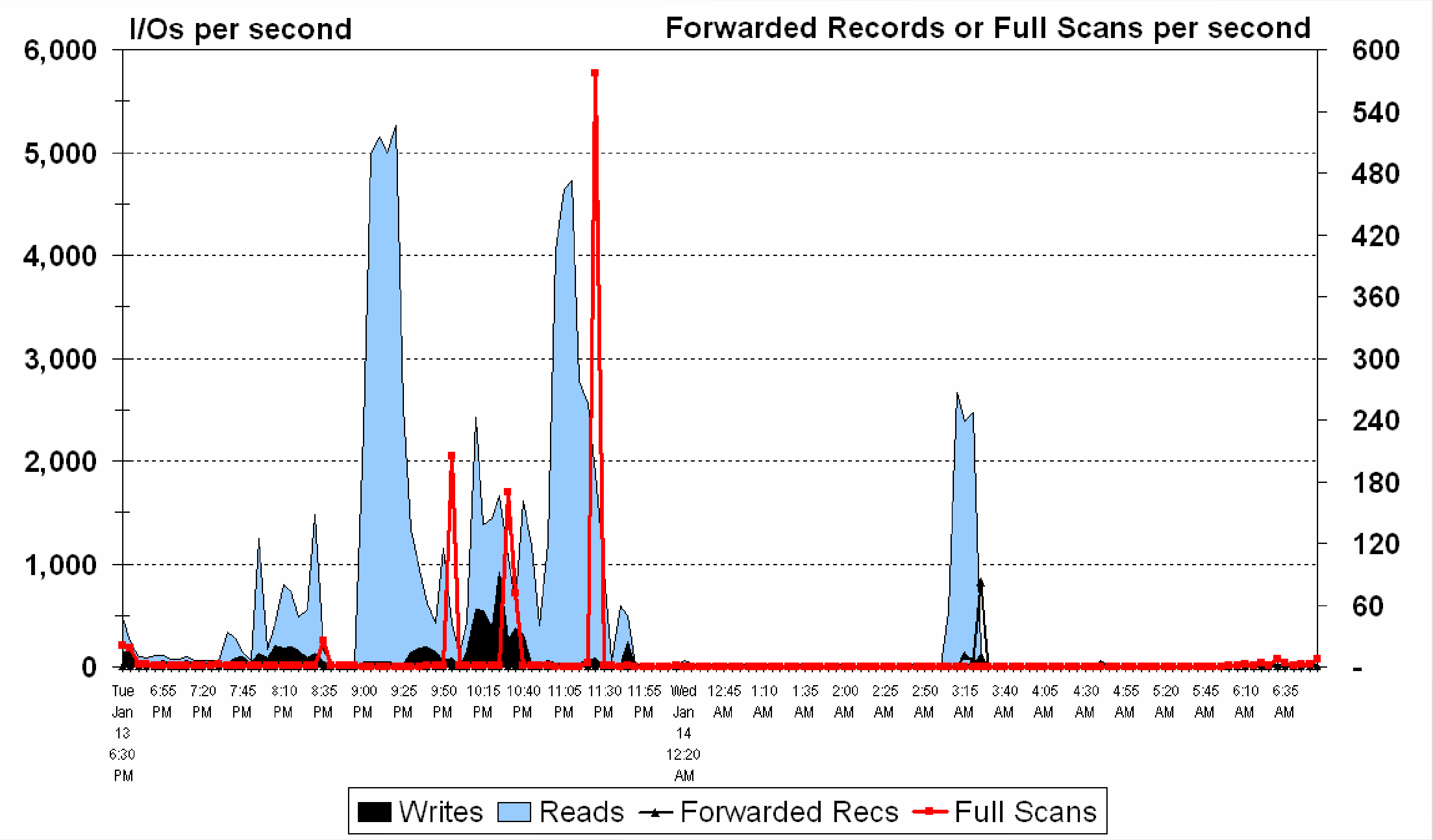


I/O Activity vs. Access Methods

- Direct graphical comparison of these entities is very helpful
- Shows whether physical and logical linear searches, i.e., forwarded records and full scans, result in physical I/Os or are completely satisfied from memory
- Many linear searches can be against very short tables that are always in memory
- Comparison distinguishes relatively harmless ones from those that impact the I/O subsystem

UNISYS
Imagine it. Done.

SQL Server I/O, Forwarded Record, & Full Scan Graph



UNISYS

Imagine it. Done.

Locks Object

- **One of the most important objects**
- ***Number of Deadlocks/sec* critical**
- **SQL Profiler can provide information about how deadlock was created**
- ***Lock Timeouts/sec* also critical**
  - # of lock requests that exceed maximum specified wait time
  - Monitors each type of lock

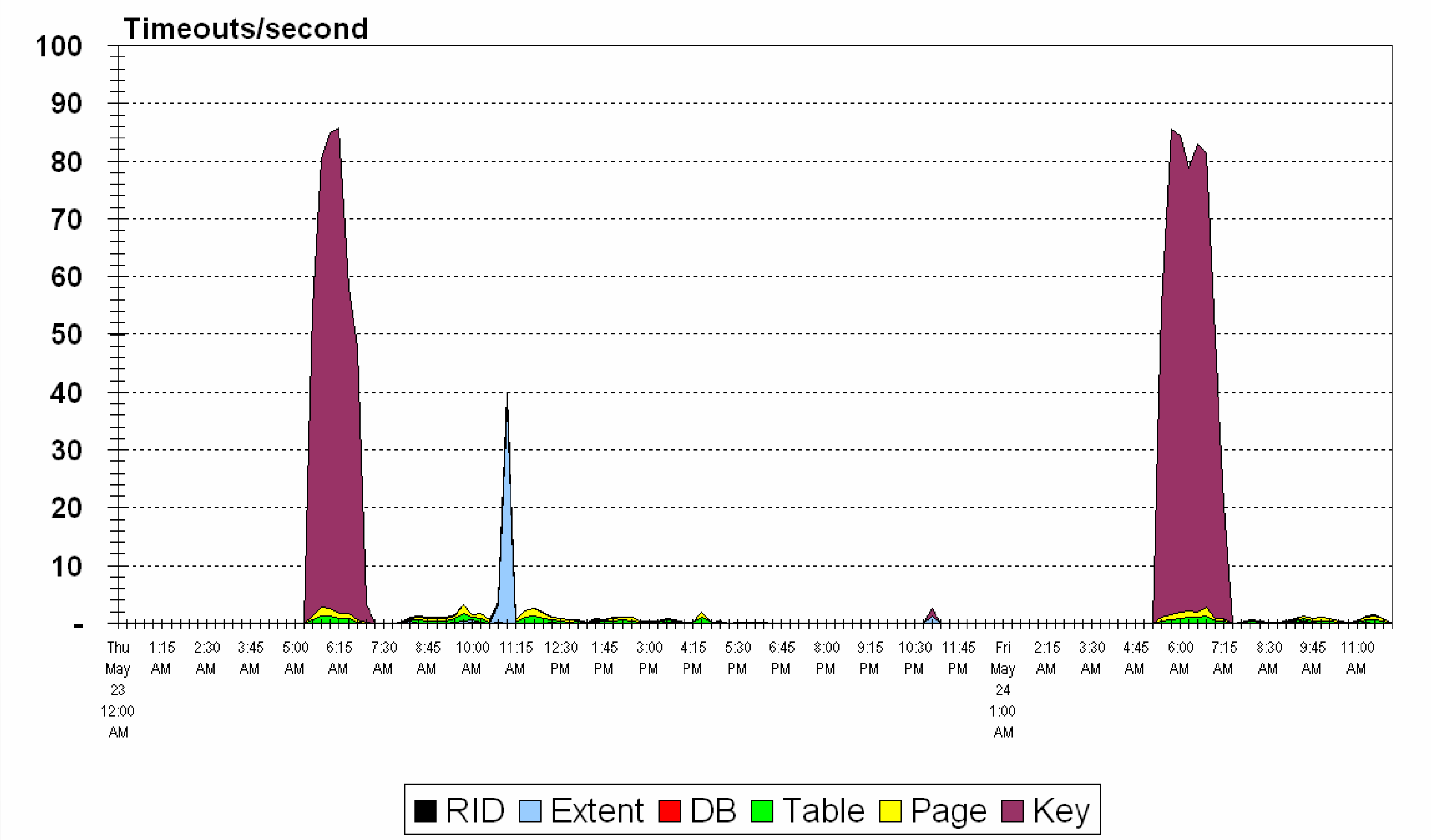
UNISYS
Imagine it. Done.

Lock Types/Instances

Item	Description
Database	Entire database
Extent	Contiguous group of 8 data pages or index pages
Key	Row lock within index
Page	8-kilobyte (KB) data page or index page
RID	Row ID. Used to lock single row within table
Table	Entire table, including all data & indices

UNISYS
Imagine it. Done.

Lock Timeouts Graph



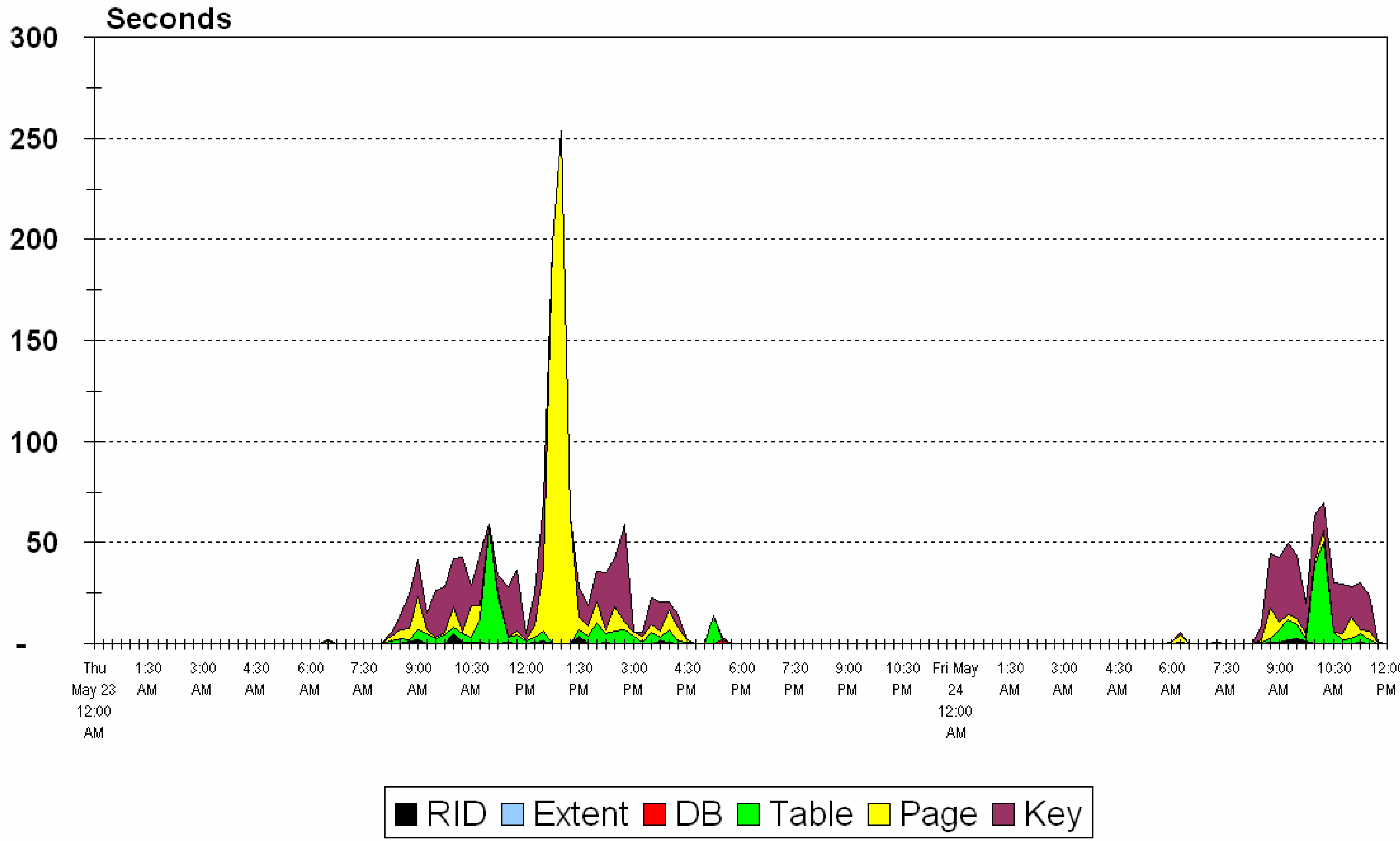
UNISYS

Imagine it. Done.

Other Lock Counters

- ***Average Wait Time (ms)***
  - Measures average time each lock request was forced to wait
- **Useful to sum these to prevent averages from disguising problems**
  - Calculate percentage of interval spent waiting
- ***Lock Waits/sec***
  - Records how often lock requests waited
- **Trace duration filter does not apply to lock timeouts**

Total Lock Wait Times

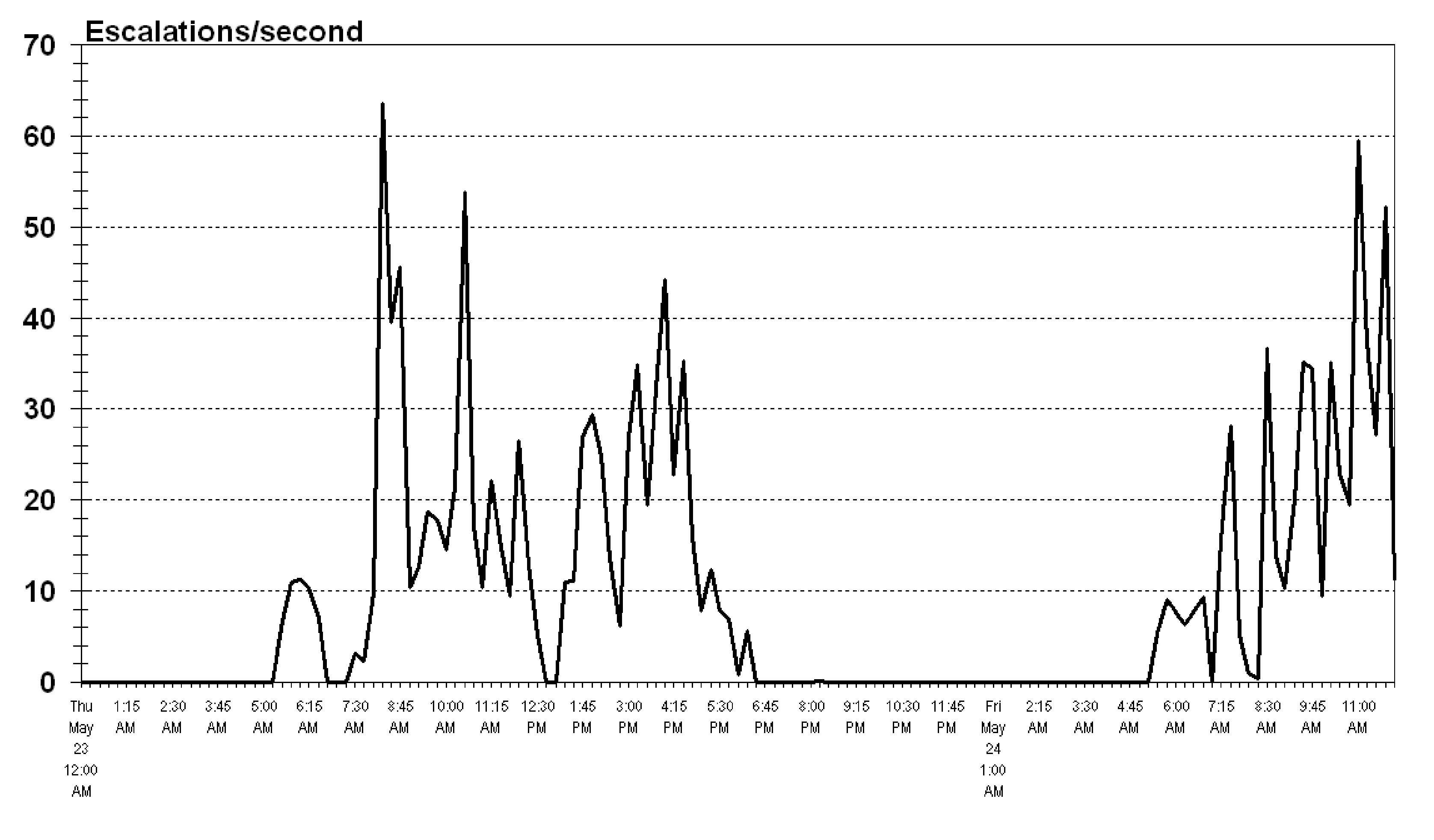


Lock Escalations

- **Row, key, or page locks automatically escalated to coarser table locks as appropriate**
  - Single table lock acquired
  - Many lower level locks released
- **Recorded in *Table Lock Escalations/sec***
- ***Lock Owner Blocks Allocated* and *Lock Blocks Allocated* can be used to validate that applications hold too many locks for too long**

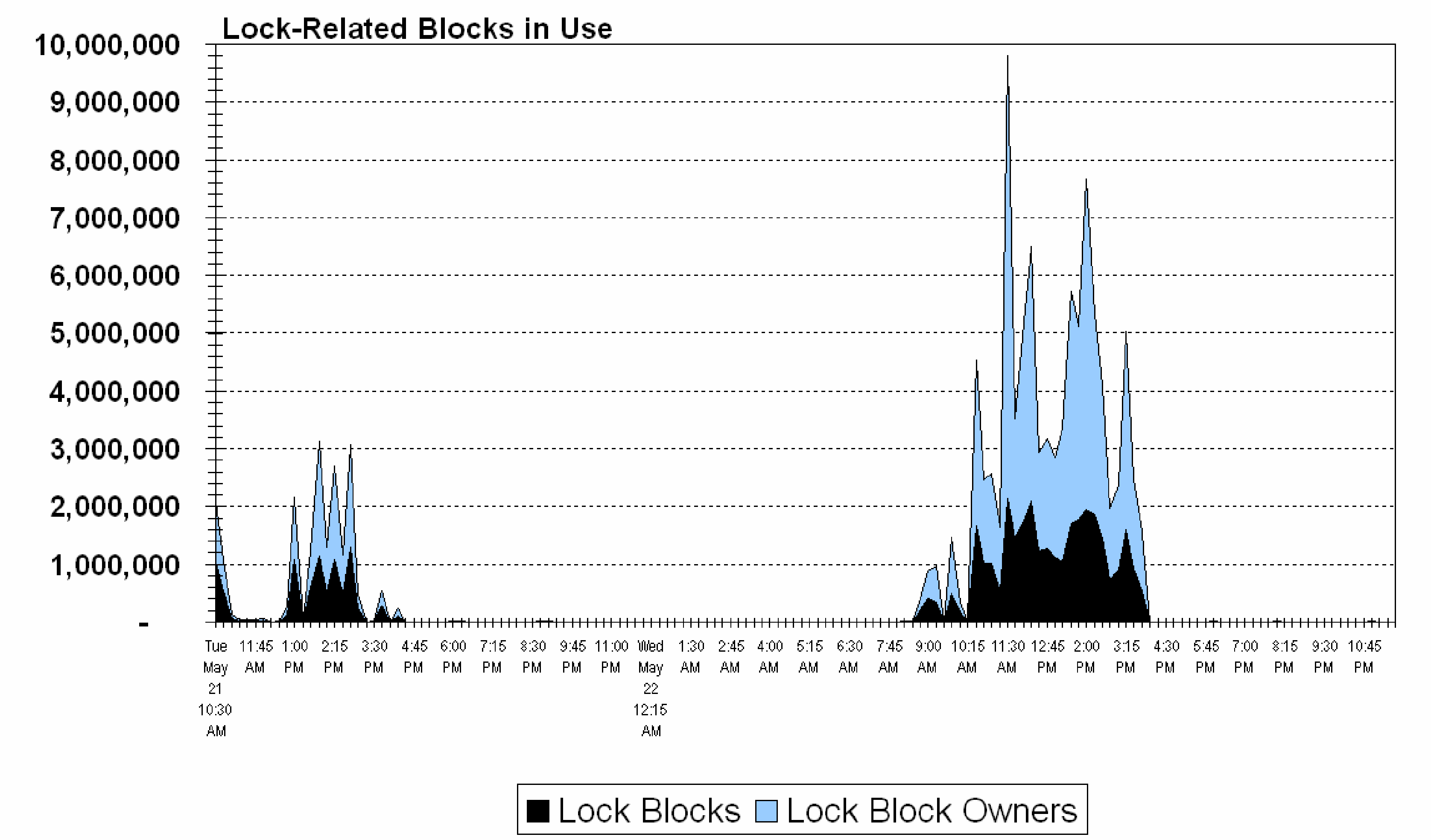
UNISYS
Imagine it. Done.

Table Lock Escalations



UNISYS
Imagine it. Done.

Lock Block & Lock Block Owners Graph



UNISYS
Imagine it. Done.

Latches

- **Latches**
  - Lightweight, short-term synchronization objects
  - Protect action that need not need be locked for life of transaction

UNISYS
Imagine it. Done.

Latch Object Counters

- **Counters**
  - *Average Latch Wait Time (ms)*
  - *Latch Waits/sec*
  - *Total Latch Wait Time (ms)*

UNISYS
Imagine it. Done.

Avg. Latch Wait Time (ms) Counter

- **Large values, e.g., greater than one second**
  - Indicate large number of physical I/Os or long I/O times
  - Check following counters
    - *Page Reads/sec* and *Page Writes/sec*
    - System PhysicalDisk object, especially *Avg. Disk Sec/Transfer*
  - Often coincide with low buffer cache hit ratios

UNISYS
Imagine it. Done.

% Disk Times & Queue Lengths

- *% Disk Times* useless because they are simply restatements of queue lengths using percent format
- Perfmon constrains these to 100%
- Queue lengths can no longer be interpreted as most Windows performance books suggest, i.e., disk is in trouble when queue length > 2
- Queue lengths of 14 or more are common, even on well-performing I/O subsystems

Physical I/O Measurements

- **Only I/O time is measured directly**
- **Disk driver provides I/O times to Windows**
- **Due to driver's location in I/O path**
  - I/O time = service time + queue time
- **May not be possible to improve large service times due to physical or financial constraints**

UNISYS
Imagine it. Done.

Large SQL Server I/Os

- Beginning with Service Pack 3, SQL Server can generate very large I/Os, e.g., larger than 65,535 bytes
- 131,070 byte and larger I/Os have been observed (see Example #3)
- HBAs can be saturated fairly quickly under these conditions
- I/O service times can cause I/O times to be high even if queuing does not occur

I/O Time Calculations

- Important to know whether queuing is causing large I/O times
- Use Little's Law to compute missing statistics

Little's Law

N = X * R

- N => average # customers at a service center
- X => program completion rate
- R => average elapsed time

UNISYS
Imagine it. Done.

Using Little's Law to Compute Missing I/O-Related Times

- **All calculations use PhysicalDisk counters**
- **Disk Utilization = 100 - *% Idle Time***
- **Disk service time = Disk Utilization / *Disk Transfers/sec***
- **Disk queue time = *Avg. Disk sec/Transfer* - Disk service time**

UNISYS
Imagine it. Done.

RAID Example Calculations #1

- Disk Utilization = 36.57%
- Disk Transfers/sec = 0.65
- Avg. Disk sec/Transfer = 2.0095
- Disk service time = .3657 / 0.65 = 0.563 seconds or 563 milliseconds
- Disk queue time = 2.0095 – 0.563 = 1.447 seconds or 1,447 milliseconds
- Bytes/Transfer = 1,307

UNISYS
Imagine it. Done.

RAID Example Calculations #2

- Disk Utilization = 77.67%
- Disk Transfers/sec = 30.89
- Avg. Disk sec/Transfer = 2.4424
- Disk service time = .7767 / 30.89 = 0.025 seconds or 25 milliseconds
- Disk queue time = 2.4424 – 0.025 = 2.4174 seconds or 2,4174 milliseconds
- Bytes/Transfer = 22,437

Imagine it. Done.

RAID Example 1 vs. 2

- I/O times are not that far apart despite being outrageously high
- Queuing being encountered for both disks
- Low I/O rate of Disk #1 appears to contribute to high service times
  - 1,307 bytes should not require 563 ms

UNISYS
Imagine it. Done.

RAID Example 1 vs. 2

- **Disk #2 is doing much more work**
  - Utilization is double
  - I/O size is 17 times larger
  - Service time is much more reasonable @ 25 ms
- **Problems began when faster processor complex attached**
- **Solution was to reconfigure EMC drives**

UNISYS
Imagine it. Done.

RAID Example Calculations #3

- Disk Utilization = 99.59%
- Disk Transfers/sec = 58.2
- Avg. Disk sec/Transfer = 0.7678
- Disk service time = 0.9959 / 58.2 = 0.0171 seconds or 17.1 milliseconds
- Disk queue time = 0.7678 – 0.0171 = 0.7507 seconds or 750.7 milliseconds
- Bytes/Transfer = 168,536

RAID Example #3 Discussion

- **100% utilization is suspicious, but RAID may be functioning well enough**
  - In this case, it obviously is not
- **17 ms service times are good considering average I/O size**
- **Queuing is the problem**
- **Drives comprising disk were clearly saturated**

RAID Example #3 Discussion

- Another disk processed 143.7 I/Os per second @ 7.8 ms per I/O
- Queue time was 2.1 ms
- Service time was 5.7 ms
- 28,786 Bytes/Transfer
- When all disks were combined, HBA was at the limit
- Solution was to add drives and HBAs

UNISYS
Imagine it. Done.

SQL Statement Handling

- **Batch**
  - Group of SQL statements
  - Possibly hundreds or thousands of lines
  - Must be parsed and compiled into an optimized execution plan
- **Compilation and parsing can be**
  - Quite resource intensive
  - Time-consuming

SQL Statistics Object

- **Most important counters**
  - *Batch Requests/sec*
  - *SQL Compilations/sec*
  - *SQL Re-Compilations/sec*
- **Use with Cache Manager object *Cache Hit Ratio* counter**

UNISYS
Imagine it. Done.

Batch Requests/sec

- **Number of select, insert, and delete statements**
- Each **of these statements triggers a batch event, which causes counter to be incremented**
- Note**: This** includes **each of these statement types that are executed within a stored procedure**

SQL Server Connection Affinity

- **Documented in Q299641**
- ***Batch Requests/sec* can be compared with System *Context Switches/sec* counter to highlight need for SQL Server Connection Affinity**
- **Network packet comparison with System *Context Switches/sec* counter also useful**
- **Processor *% DPC Time* counter can also be useful in this endeavor**

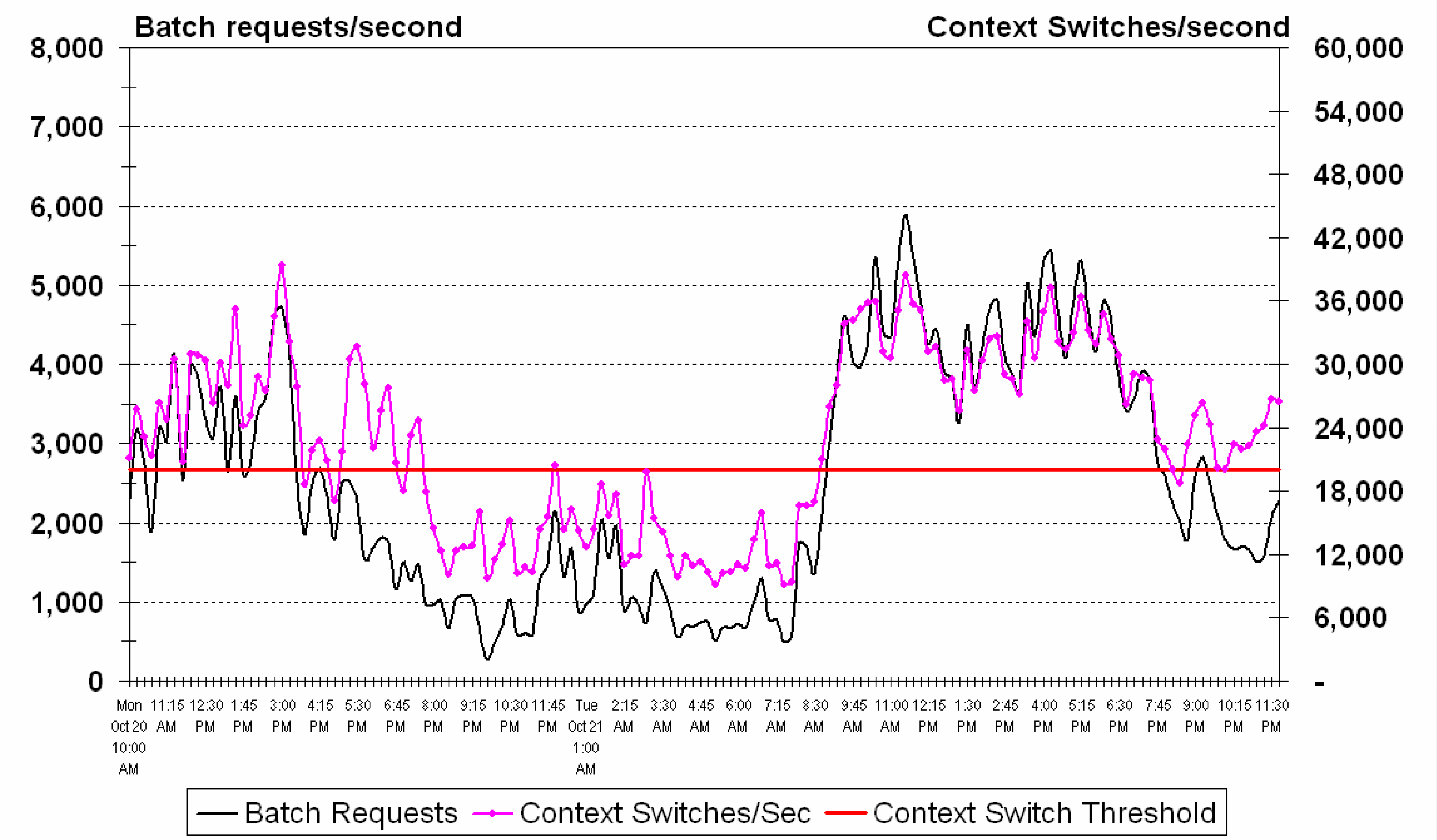
UNISYS
Imagine it. Done.

Connection Affinity Example

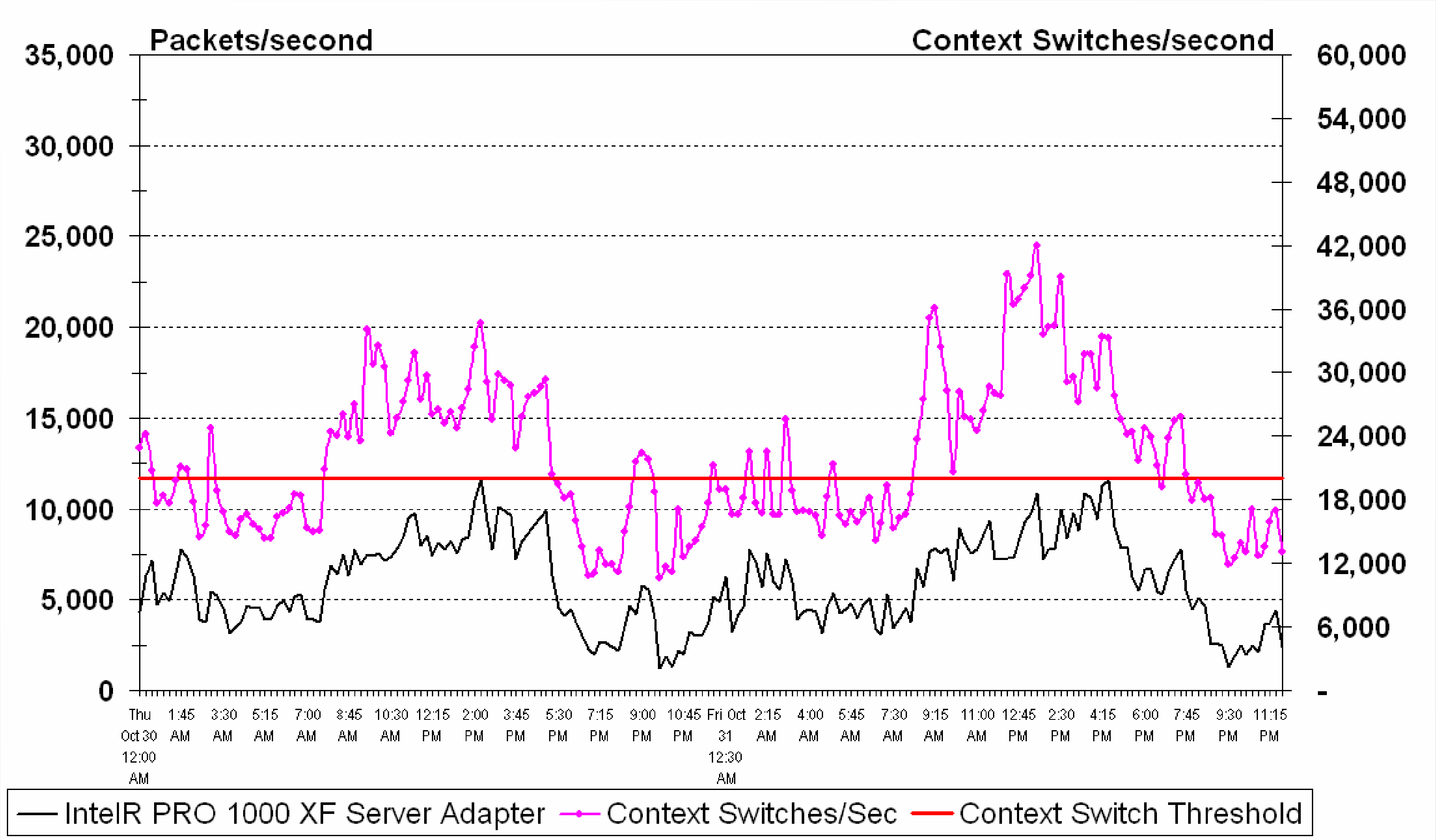
- *Batch requests/sec* correspond almost exactly with System *Context Switches/sec*
- Network packet traffic also almost perfectly matches System *Context Switches/sec*
- Processor DPC activities correspond closely as well

UNISYS
Imagine it. Done.

Batch Requests vs. Context Switching Graph



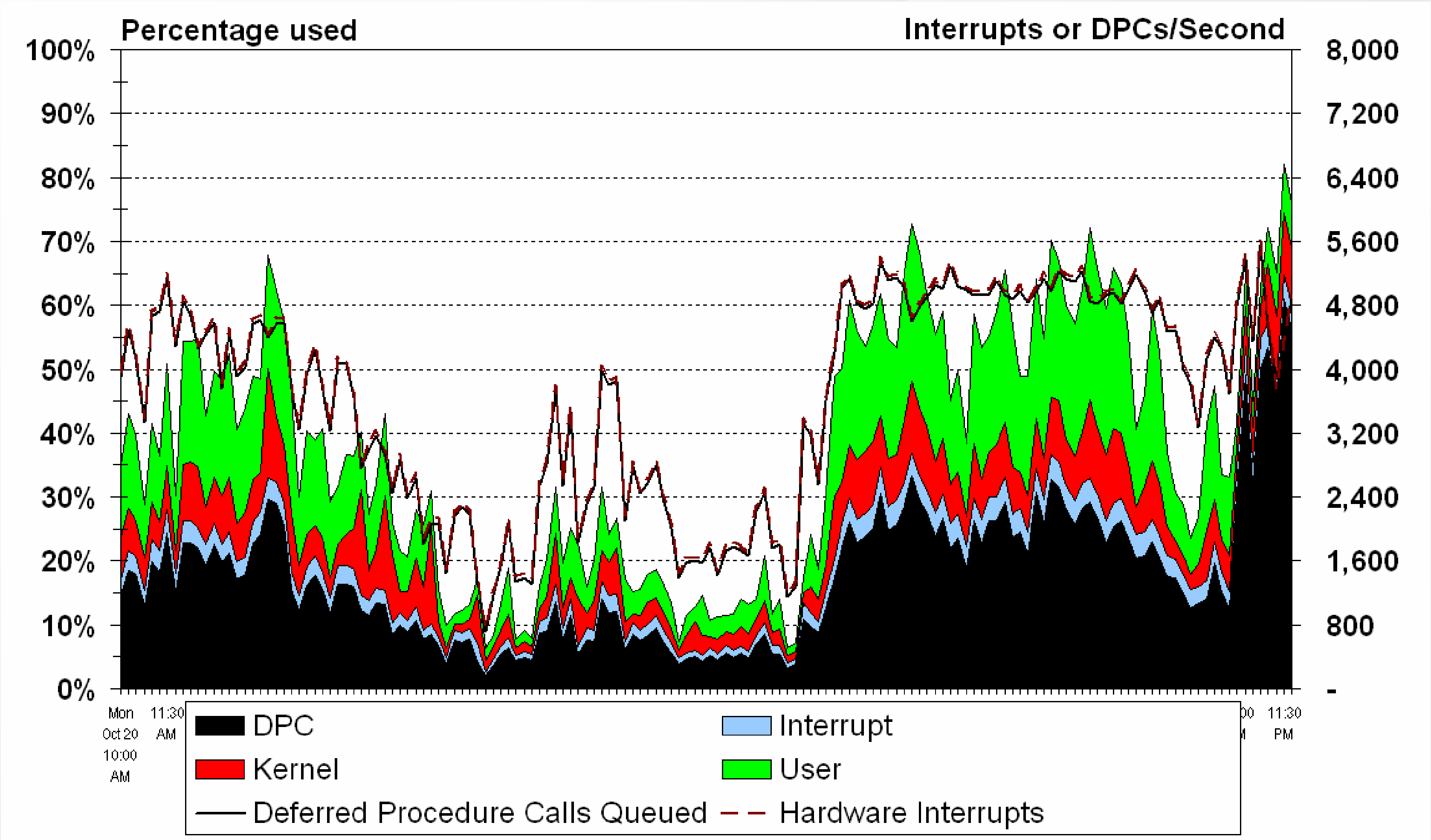
Network Card Packet Traffic & Context Switches Graph



UNISYS

Imagine it. Done.

Processor Overview Graph



UNISYS
Imagine it. Done.

Stored Procedure Compilation

- **Allows batch to be parsed and compiled only once (hopefully)**
- **Execution plan cached & re-used unless**
  - Removed from cache
  - Execution plan invalidated because of database changes
- **If stored procedure requested after removal or invalidation, it is recompiled**

UNISYS
Imagine it. Done.

Cache Manager Object

- **Monitors various execution-related entities and their re-use**
  - Prepared SQL plans
  - Procedure plans
  - Trigger plans
  - Normalized trees
- **Used in SQL statement, stored procedure and trigger compilation, optimization, and execution**

UNISYS
Imagine it. Done.

Cache Hit Ratio

- **Most important**
- **Should be 90% or higher**
- **Lower values**
  - Indicate too many ad-hoc queries
  - Often associated with higher values of
    - *SQL Compilations/sec* (SQL Statistics object)
    - *SQL Re-Compilations/sec*

UNISYS
Imagine it. Done.

Guidelines

- ***SQL Compilations/sec*** **should be less than 40% of** ***Batch Requests/sec***
- **High compilation rates frequently**
  - Correspond with lower Cache Manager cache hit ratios
  - Indicate lack of stored procedure usage
  - Indicate possible memory shortage

UNISYS
Imagine it. Done.

General Statistics Object

- **Useful in capacity planning situations**
- ***Logins/sec***
- ***Logouts/sec***
- ***User Connections***
- **Useful in calculating work per user or connection**

UNISYS
Imagine it. Done.

Databases Object

- Each database is a performance counter instance
- Log and transaction counters most important

Imagine it. Done.

Log

- **Database journal**
- **Used for recovery**
- **Changes written here before database**
- **Can dramatically hinder database performance if placed on busy disk**
- **Should be on own disk volume**
  - Minimizes disk head movement

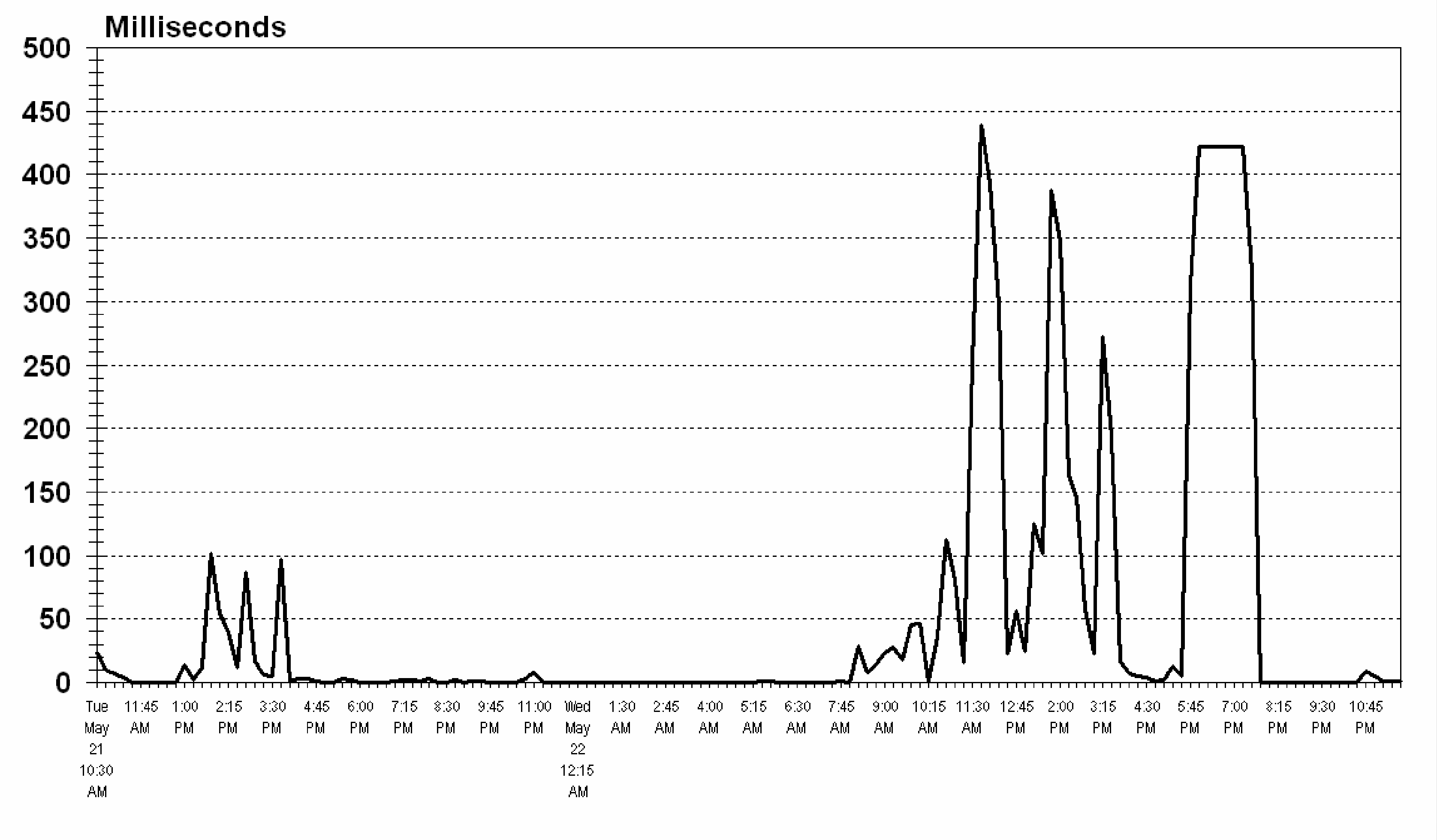
UNISYS
Imagine it. Done.

Log Flush Wait Time Counter

- Measures total time database commits waited for log flushes
- Obviously, should be small

UNISYS
Imagine it. Done.

Average Log Write Waiting Times Graph



UNISYS
Imagine it. Done.

Potentially Confusing Log Counters

- ***Log Flushes/sec*** **measures number of log buffers flushed to disk**
- ***Log Flush Waits/sec*** **measures number of flushes that had to wait**
  - Seems like an ideal number
- **Waits should be subset of total?**
  - Unfortunately, only in some cases

UNISYS
Imagine it. Done.

Recovery Models

- **Full Recovery**
  - Every database change logged
  - Recover to last complete transaction
- **Bulk-logged**
  - Bulk operations minimally logged
  - Recover to end of transaction log backup
- **Simple**
  - Recover to last database backup
- **Waits a subset of Total only for Simple model, which is used least in production**

Other Database Counters

- *Transactions/sec* counter indicates which databases updated most frequently
- Particularly important because all Tempdb transactions are monitored

UNISYS
Imagine it. Done.

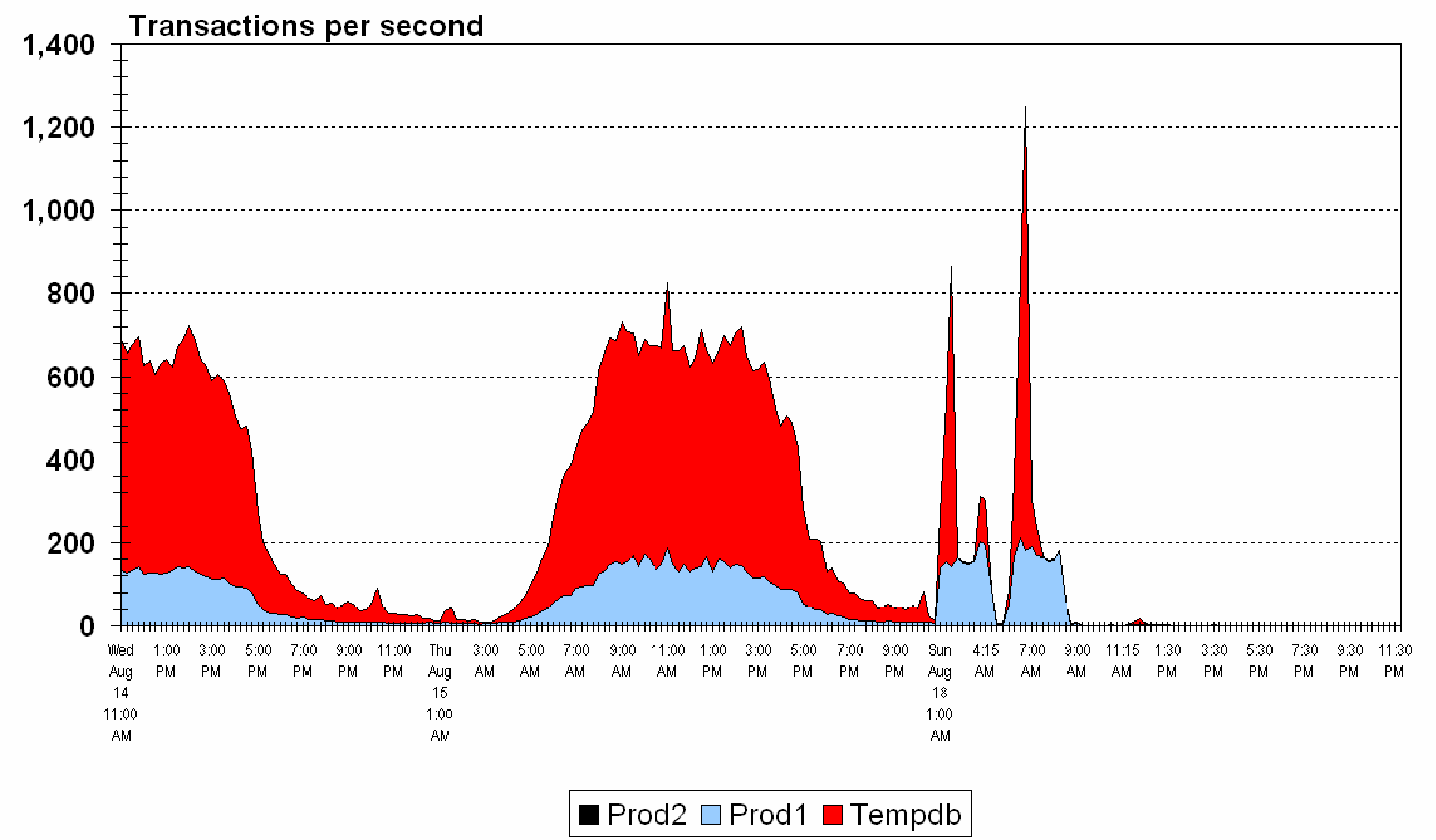
Tempdb

- Contains all temporary disk tables and workspaces
- Overuse can significantly hinder scalability
- Can become major bottleneck
- Use creatively designed queries to reduce Tempdb activity

Table Variables

- More efficient than pure tables
- Unfortunately, still use Tempdb as other temporary tables do
- Obviously, spreading Tempdb across several physical disks helps performance
- Not so obviously, increasing number of physical Tempdb files can reduce file access bottleneck, especially if Tempdb hit very hard

Database Transaction Volumes Graph



UNISYS

Imagine it. Done.

Other Database Counters

- ***Bulk Copy Throughput/sec***
  - Useful for monitoring efficiency and frequency of flat file loads into database tables
- **Bulk copies/inserts can *easily* cause table escalation and locking**
- **Extremely efficient method for mass data loads**
- **Should be infrequent during online day**

Statistics

- Computed for tables and indices
- Enables query optimization
- Can be expensive to create or update depending upon sample size and frequency
- May want to update these manually during off-peak times instead of using automatic defaults
- *DBCC Logical Scan Bytes/sec* useful for detecting when statistics recalculated

Statistics

- **Set on by default in Tempdb**
- **Application Sentinel SQL Optimizer detects this**
- **Various options can be used to control impact of statistics update or recreation**
  - FULLSCAN
  - SAMPLE <n> PERCENT or ROWS
  - RESAMPLE
  - ALL or COLUMNS or INDEX

UNISYS
Imagine it. Done.

Database Size Counters

- Log File(s) Size (KB)
- Data File(s) Size (KB)
- Log File(s) Used Size (KB)
- Percent Log Used
- Log Growths
- Log Truncations
- Log Shrinks

UNISYS
Imagine it. Done.

Database Size Counters

- **Useful for determining**
  - Volatility of log files
  - Frequency of database and log expansion
  - Overall sizes of databases and their logs
- **Minimize frequency of database and log expansions**

UNISYS
Imagine it. Done.

SQL Profiler

- **Bad reputation as a resource hog and performance killer need not be deserved**
- **Excessive resource consumption caused by**
  - Requesting entities that are changed constantly, e.g., locks and scans
  - Not using duration filter
  - Requesting too many entities
- **Updating GUI on monitored machine**

UNISYS
Imagine it. Done.

Events

- **Entities that are monitored**
- **Careful use of templates can** greatly **reduce resource consumption**
- **Unfortunately, duration filter does not control Lock:Timeout event logging**
  - Most records returned will be zero duration
- **Lock:Deadlock and Lock:Deadlock Timeout still valuable events**

Starting Events

- **Following events usually unnecessary because start time can be calculated from ending records using duration**
  - Stored Procedures event class
    - SP:Starting, SP:StmtStarting, RPC:Starting
  - T-SQL event class
    - SQL:BatchStarting, SQL:StmtStarting

Other Events

- **Performance events**
  - Potentially very useful
  - Execution Plan, Show Plan All, Show Plan Statistics, and Show Plan Text difficult to decipher
- **Errors and Warnings events**
  - **Extremely** useful for highlighting inefficient sorts, missing statistics, inefficient joins
  - Low overhead

Warnings Summary

Database ID	Object ID	Event Name	Event SubClass Name	Integer Data	Count
1		**Sort Warnings**	**Single pass**		354
6		**Sort Warnings**	**Single pass**		9,006
6		**Sort Warnings**	**Multiple pass**		1,323
6	3	**Hash Warning**	**Hash recursion**	0	540
6	3	**Hash Warning**	**Hash recursion**	1	6
6	5	**Hash Warning**	**Hash recursion**	0	219
6	7	**Hash Warning**	**Hash recursion**	0	1,062
6	11	**Hash Warning**	**Hash recursion**	0	27,477
8		**Sort Warnings**	**Single pass**		153

UNISYS
Imagine it. Done.

sp_trace Instead of Profiler

- Most efficient to use sp_trace commands to capture trace information because no GUI involved
- Better than using remote Profiler because session does not restart after network interruption

UNISYS
Imagine it. Done.

sp_trace Commands

- sp_trace_create defines a trace, but does not start it
- sp_trace_setevent “adds or removes an event or event column to a trace”
- sp_trace_setfilter “applies a filter to a trace”
- sp_trace_setstatus “modifies the current state of the specified trace,” e.g., starts or stops trace

sp_trace

- **Can be used to “sample” trace data instead of continuously capturing data**
  - Collect for a few minutes and then stop
  - Restart collection at some future point
- **Useful on high-volume systems where any tracing could be noticed**
- **Job can be set up to implement this**

Using SQL Server to Analyze Traces

- **Traces can be imported easily into a SQL Server database using**
  - T-SQL commands
  - SQL Profiler
- **Stored Procedures can be used to**
  - Summarize data
  - Replace numeric IDs with meaningful text
  - Locate offending queries
  - Join trace data with other performance data

UNISYS
Imagine it. Done.

Conclusions

- Many performance counters available with SQL Server
- Several have useful descriptions associated with them, but many do not
- Most objects and counters pertain to SQL Server as an entity without regard to a specific user, query, or database

UNISYS
Imagine it. Done.

Conclusions

- When complex applications access multiple databases under one SQL Server instance, PerfMon counters alone do not provide enough information
- Extremely important to combine SQL Server performance information with system performance information, especially processor, memory, and I/O

Conclusions

- **SQL Query and SQL Profiler both provide extremely useful insights into how specific databases, queries, transactions, batches, and stored procedures perform**

Conclusions

- Many analysts believe that SQL Profiler cannot be run against a production system without severely damaging performance
- This need not be true!
- Lightweight Profiler templates or trace T-SQL routines can be used to gather very specific and inexpensive information regularly

Conclusions

- SQL Trace output can be imported into a SQL Server database for fast and easy analysis
- Once specific queries or stored procedures have been identified as offenders, additional data can be gathered for just those entities

References

- Kalen Delaney, *Inside Microsoft SQL Server 2000*
- Mark Friedman and Odysseas Pentakalos, *Windows 2000 Performance Guide*
- *Microsoft SQL Server 2000 Books Online*

UNISYS
Imagine it. Done.

References

- Edward Whalen, Marcilina Garcia, Steve DeLuca, and Dean Thompson, *Microsoft SQL Server 2000 Performance Tuning*

UNISYS
Imagine it. Done.