# Wait-Time Based Oracle Performance Management

Dallas Deeds

Nationwide

# Who am I?

- DBA for Nationwide Insurance for 12 years
- Specializing in performance optimization using Oracle Wait Interface and OS utilities since 2001
- Responsible for performance of 650 Oracle databases at Nationwide

# The shop

8 Production DBAs

650 Oracle databases (and growing)
- sizes ranging from 1 GB to 6 TB

Primarily on Solaris machines
- A sprinkling of HP, AIX, Linux

Primarily frame storage

# Agenda

- Wait-Based Strategies and Tools
- Case Study One: Hot Block Issue
- Case Study Two: Full Table Scans
- Case Study Three: Inefficient Indexes
- Case Study Four: CBC Latching
- Q&A

# Working the Right Problems?

- After spending an agonizing week tuning Oracle to minimize I/O operations, management typically rewards you with:

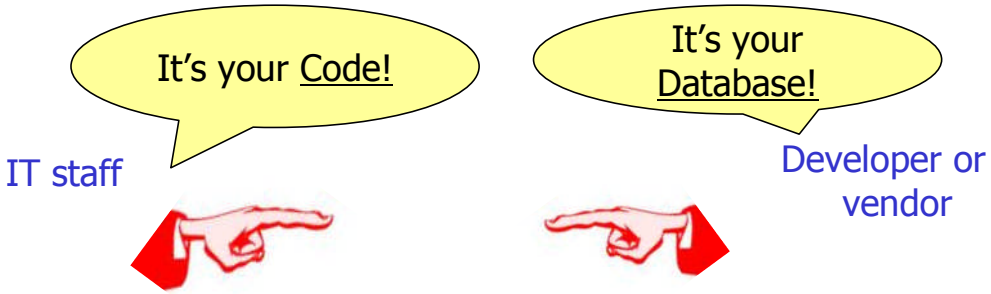
  - A. An all expense paid vacation
  - B. A free lunch
  - C. Crumbs from the kitchen

  - D. Reward? Nobody even noticed!
  - E. You slacker DBA! Don't you ever do any work?

# Why Does This Happen?

- Many tools measure system health
- Assumption:  If I make the database healthy, users benefit
- Symptoms
  - DBA finds "big" problem and fixes it, users report no impact
  - Lots of data to review and things to fix, not sure which to do first
  - Unclear view of performance leads to Finger-pointing

| Statistic | Total |
|---|---|
| CPU used by this session | 2,317,384 |
| CPU used when call started | 2,317,379 |
| CR blocks created | 36,901 |
| DBWR checkpoint buffers written | 101,603 |
| DBWR checkpoints | 36 |
| DBWR transaction table writes | 1,046 |
| DBWR undo block writes | 44,953 |
| SQL*Net roundtrips to/from client | 18,550,671 |
| active txn count during cleanout | 38,735 |
| background checkpoints completed | 36 |
| background checkpoints started | 36 |
| background timeouts | 7,234 |
| branch node splits | 7 |
| buffer is not pinned count | 78,660,488 |
| buffer is pinned count | 94,826,641 |
| bytes received via SQL*Net from c | 1,109,332,214 |
| bytes sent via SQL*Net to client | 665,131,799 |
| calls to get snapshot scn: kcmgss | 49,751,137 |
| calls to kcmgas | 432,277 |
| calls to kcmgcs | 15,037 |
| change write time | 15,859 |
| cleanout - number of ktugct calls | 42,989 |



# Confio Performance Intelligence

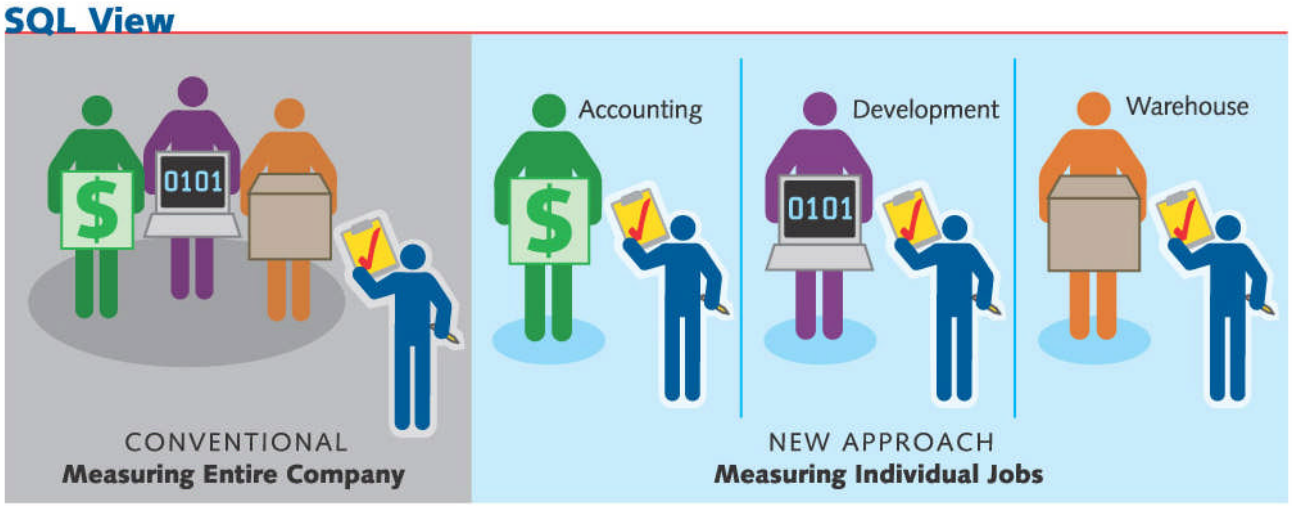
- **Three Key Principles**

1. ***SQL View***: All statistics at SQL statement level
2. ***Full View***: Separately measure every resource (Oracle wait events) to isolate source of problems
3. ***Time View***: Measure Time, not number of times a resource is utilized

# Focus on User Response Time

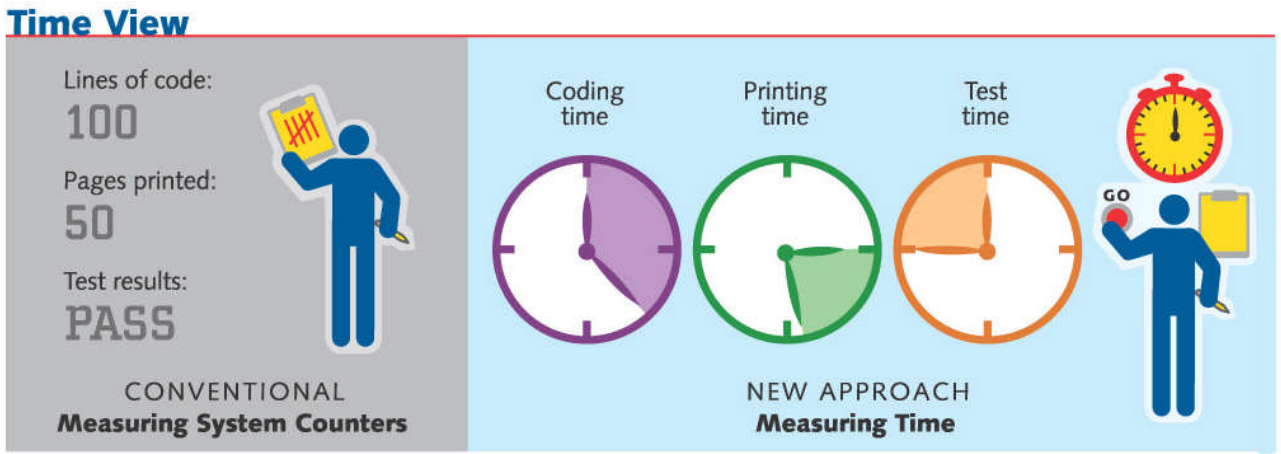
1. Identify each bottleneck affecting the user
2. Rank bottlenecks by user impact
3. Set correct expectations on impact of fix
4. Implement proven suggestions
5. Show proof the fix helped users

# SQL View Principle



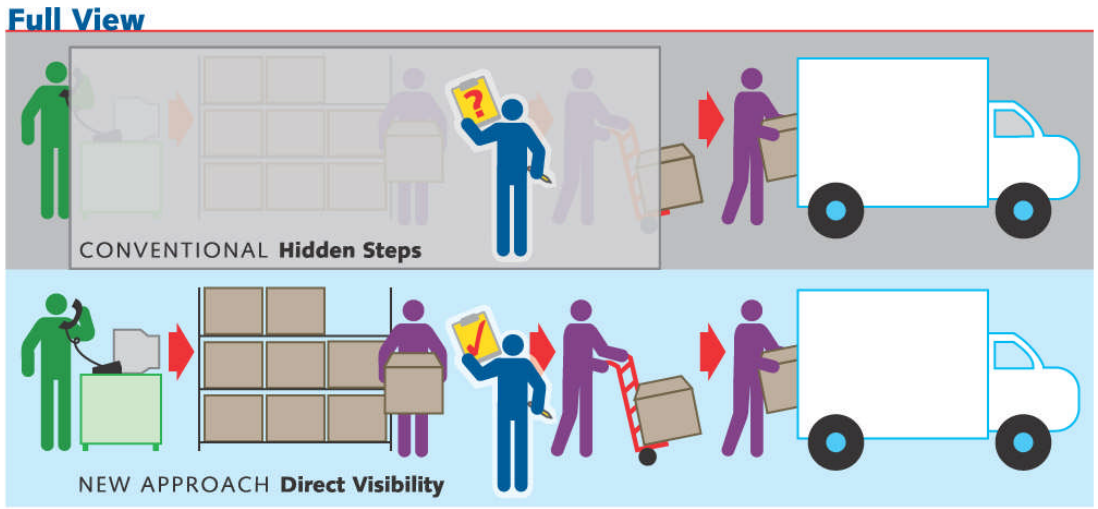

- Example: 'CEO' measuring 'employee' output
- Averaging over entire company gives no useful data
- Must measure each job separately
- DBA must manage database similarly
- Measure and identify bottlenecks for each SQL independently

# Time View Principle




- Example: 'CEO' counting 'tasks' vs. 'time to complete'
- Counting system statistics not meaningful
- Must measure Time to complete
- System stats (buffer size, hit ratios, I/O counts) do not identify where database customers are waiting
- Identify and optimize Wait Time for each SQL to optimize response time

# Full View Principle




- Example: 'CEO' measuring results with blind spot hiding key processes
- Without direct visibility, valuable info is lost
- Must have visibility to every process step
- Identify and measure each Oracle resource for each distinct SQL

# Compliant Performance Tool Types

**Two Primary Types of Tools**

- Session Specific Tools
  - Tools that focus on one session at a time by profiling 10046 trace data obtained by tracing the process
  - Examples: Hotsos Profiler, OraSRP Profiler (open source), tkprof
- Continuous DB Wide Monitoring Tools
  - Tools that focus on all sessions by sampling Oracle
  - Examples: Confio Ignite, Precise i3
- Both tools have a place in the organization

# Tracing

- Proper targeting and scoping for required for best results
  - 95 out of 100 users are running well
  - 5 out of 100 have spent 99% of time waiting for locked rows
  - If you trace one of the "95" sessions
    - No locking problems at all
    - May spend time trying to optimize the wrong thing
  - If you trace one of the "5" sessions
    - Severe locking problems
    - Appears that you could fix the locking problems and reduce user response time by 99%

# Tracing - Scoping

- Proper scoping of user actions is required to get the best data
  - Stop and start tracing to collect trace data for only the user actions you are interested in
  - Start trace when Bob mashes the ‘enter’ key
  - Stop trace when Bob’s results paint on the screen
  - Otherwise you will introduce irrelevant data
    - SQL*Net messaging
    - User actions that are not part of the problem
    - Response time gets attributed to the wrong things

# Tracing (cont)

- Advantages
  - Very precise – accurate to the microsecond level (depending on platform)
  - The only way to get certain performance data
  - Bind values are available
  - Provides detailed analysis even deeper than wait events
- Disadvantages
  - Only works if a known problem is going to occur in the future or if the problem is repeatable
  - Difficult to see trends over time

# Profiled trace data

**1.1. Profile by Subroutine**

The following table shows the decomposition of total task response time by internal subroutine.

| | Subroutine | Duration | | Cumulative duration | | Call count | Duration per call (seconds) | | | | Drill-down |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | seconds | % *R* | seconds | % *R* | | mean | min | skew | max | |
| 1. | **db file sequential read** | **2,573.170** | **96.4%** | 2,573.170 | 96.4% | **711,712** | 0.003615 | 0.000018 | | 1.515752 | SQL |
| 2. | CPU service, execute calls | 168.950 | 6.3% | 2,742.120 | 102.7% | 80 | 2.111875 | 0.000000 | | 168.930000 | SQL |
| 3. | log file switch completion | 0.358 | 0.0% | 2,742.478 | 102.7% | 4 | 0.089602 | 0.009283 | | 0.274950 | |
| 4. | db file parallel read | 0.038 | 0.0% | 2,742.516 | 102.7% | 1 | 0.037590 | 0.037590 | | 0.037590 | |
| 5. | unaccounted-for between dbcalls | 0.033 | 0.0% | 2,742.549 | 102.7% | 5 | 0.006675 | 0.000085 | | 0.032664 | |
| 6. | log file sync | 0.023 | 0.0% | 2,742.572 | 102.7% | 1 | 0.022776 | 0.022776 | | 0.022776 | |
| 7. | CPU service, fetch calls | 0.020 | 0.0% | 2,742.592 | 102.7% | 1,121 | 0.000018 | 0.000000 | | 0.010000 | |
| 8. | SQL*Net message from client | 0.010 | 0.0% | 2,742.602 | 102.7% | 8 | 0.001310 | 0.000740 | | 0.002808 | |
| 9. | SQL*Net message to client | 0.000 | 0.0% | 2,742.602 | 102.7% | 8 | 0.000003 | 0.000000 | | 0.000019 | |
| 10. | CPU service, prepare calls | 0.000 | 0.0% | 2,742.602 | 102.7% | 75 | 0.000000 | 0.000000 | | 0.000000 | |
| 11. | unaccounted-for within dbcalls | -73.127 | -2.7% | 2,669.476 | 100.0% | 1,275 | -0.057354 | -73.140140 | | 0.003721 | |
| 12. | Total | 2,669.476 | 100.0% | | | | | | | | |

# Trace showing response time skew

- Out of 711,712 I/Os, 11% accounted for 72% of the response time (!)

Skew pictogram: 

| | Range {*min* ≤ *e* < *max*} (seconds) | | | Duration | | | Cumulative duration | | |
|---|---|---|---|---|---|---|---|---|---|
| | min | max | Call count | seconds | % | % *R* | seconds | % | % *R* |
| 1. | 0.000000 | 0.000001 | 0 | 0.000 | 0.0% | 0.0% | 0.000 | 0.0% | 0.0% |
| 2. | 0.000001 | 0.000010 | 0 | 0.000 | 0.0% | 0.0% | 0.000 | 0.0% | 0.0% |
| 3. | 0.000010 | 0.000100 | 10,804 | 0.412 | 0.0% | 0.0% | 0.412 | 0.0% | 0.0% |
| 4. | 0.000100 | 0.001000 | 514,859 | 184.316 | 7.2% | 6.9% | 184.727 | 7.2% | 6.9% |
| 5. | 0.001000 | 0.010000 | 105,632 | 460.772 | 17.9% | 17.3% | 645.499 | 25.1% | 24.2% |
| 6. | **0.010000** | **0.100000** | **79,157** | **1,657.105** | **64.4%** | **62.1%** | 2,302.604 | 89.5% | 86.3% |
| 7. | 0.100000 | 1.000000 | 1,254 | 263.837 | 10.3% | 9.9% | 2,566.441 | 99.7% | 96.1% |
| 8. | 1.000000 | 10.000000 | 6 | 6.728 | 0.3% | 0.3% | 2,573.170 | 100.0% | 96.4% |
| 9. | 10.000000 | 100.000000 | 0 | 0.000 | 0.0% | 0.0% | 2,573.170 | 100.0% | 96.4% |
| 10. | 100.000000 | 1,000.000000 | 0 | 0.000 | 0.0% | 0.0% | 2,573.170 | 100.0% | 96.4% |
| 11. | 1,000.000000 | Infinity | 0 | 0.000 | 0.0% | 0.0% | 2,573.170 | 100.0% | 96.4% |
| 12. | | Total | 711,712 | 2,573.170 | 100.0% | 96.4% | | | |

# Response time skew

- 10046 trace data allows the performance analyst to illustrate things that are difficult to see otherwise
- Like showing how many I/Os took between .034 and .1 seconds
  - By datafile
  - With minimum, maximum and average times
- This sort of data is handy when arguing with your disk folks

# Trace profile example – skewed reads

**Histogram**

| | $p1 | Duration (seconds) | | Cumulative duration (seconds) | | Call count | Duration per call (seconds) | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | mean | min | max |
| 1. | 108 | 40.257443 | 9.0% | 40.257 | 9.0% | 793 | 0.050766 | 0.034011 | 0.096975 |
| 2. | 109 | 36.370212 | 8.1% | 76.628 | 17.1% | 714 | 0.050939 | 0.034039 | 0.099068 |
| 3. | 107 | 33.373796 | 7.5% | 110.001 | 24.6% | 650 | 0.051344 | 0.034015 | 0.099882 |
| 4. | 111 | 32.765686 | 7.3% | 142.767 | 31.9% | 656 | 0.049948 | 0.034026 | 0.099371 |
| 5. | 112 | 30.900416 | 6.9% | 173.668 | 38.9% | 608 | 0.050823 | 0.034013 | 0.099653 |
| 6. | 110 | 27.729186 | 6.2% | 201.397 | 45.1% | 541 | 0.051255 | 0.034013 | 0.099906 |
| 7. | 113 | 26.951323 | 6.0% | 228.348 | 51.1% | 526 | 0.051238 | 0.034005 | 0.099154 |
| 8. | 114 | 25.558787 | 5.7% | 253.907 | 56.8% | 503 | 0.050813 | 0.034035 | 0.099987 |
| 9. | 22 | 21.874214 | 4.9% | 275.781 | 61.7% | 415 | 0.052709 | 0.034003 | 0.099381 |
| 10. | 11 | 21.325552 | 4.8% | 297.107 | 66.5% | 404 | 0.052786 | 0.034084 | 0.096402 |
| 11. | 21 | 19.793488 | 4.4% | 316.900 | 70.9% | 364 | 0.054378 | 0.034134 | 0.095490 |
| 12. | 20 | 16.061555 | 3.6% | 332.962 | 74.5% | 308 | 0.052148 | 0.034139 | 0.096461 |
| 13. | 16 | 14.803718 | 3.3% | 347.765 | 77.8% | 288 | 0.051402 | 0.034024 | 0.097290 |
| 14. | 18 | 14.749035 | 3.3% | 362.514 | 81.1% | 280 | 0.052675 | 0.034042 | 0.096594 |
| 15. | 15 | 14.615227 | 3.3% | 377.130 | 84.4% | 279 | 0.052384 | 0.034022 | 0.097969 |
| 16. | 12 | 13.650224 | 3.1% | 390.780 | 87.4% | 262 | 0.052100 | 0.034053 | 0.099772 |
| 17. | 10 | 13.286173 | 3.0% | 404.066 | 90.4% | 249 | 0.053358 | 0.034019 | 0.096539 |
| 18. | 13 | 12.073496 | 2.7% | 416.140 | 93.1% | 228 | 0.052954 | 0.034269 | 0.095813 |
| 19. | 17 | 10.866008 | 2.4% | 427.006 | 95.5% | 211 | 0.051498 | 0.034002 | 0.097333 |
| 20. | 14 | 10.020653 | 2.2% | 437.026 | 97.8% | 197 | 0.050866 | 0.034193 | 0.094601 |
| 21. | 19 | 9.975718 | 2.2% | 447.002 | 100.0% | 187 | 0.053346 | 0.034089 | 0.099326 |
| 22. | Total | 447.001910 | 100.0% | | | 8,663 | 0.051599 | 0.034002 | 0.099987 |

# Continuous DB Wide Monitoring Tools

- 24/7 sampling provides real-time and historical perspective
- Allows DBA to go back in time
  - User calls, says the batch flow was hung at 3 AM this morning
- Use built-in utilities - trend reports, graphs, etc to communicate with other groups
  - What things are starting to perform poorly?
  - What progress have we made while tuning?
  - When did the code change go in that is now thrashing the system to death?

# Oracle Wait Events

- Oracle instruments more of the kernel with each release
- Often expanding events (like TX Enqueue)
- 379 wait events in 9iR2
- 871 wait events in 10GR2
- 928 wait events in 11.1

# Oracle Wait Interface

- V$SESSION_WAIT (X$KSUSECST)
  - SID (join to v$session)
  - EVENT
  - P1, P1RAW, P2, P2RAW, P3, P3RAW
  - STATE = 'WAITING' – currently waiting on event
  - STATE = 'WAITED…' – currently on CPU (or in queue)
- Oracle 10g added this info to V$SESSION

# Oracle Sessions

- V$SESSION (X$KSUSE)
  - SID
  - USERNAME
  - SQL_HASH_VALUE
    - Join to V$SQL
  - PROGRAM
  - MODULE / ACTION
    - DBMS_APPLICATION_INFO
  - PLAN_HASH_VALUE
    - Join to V$SQL_PLAN

# Base Query

```
SELECT
   sid, username, program, module, action,
   machine, osuser, sql_hash_value, …
   decode(state, 'WAITING', event, 'CPU') event,
   p1, p1raw, p2, …,
   SYSDATE
FROM V$SESSION s
WHERE s.status = 'ACTIVE'
AND event NOT IN (<idle wait events>);
```

# Additional Information

- V$SESSION
  - service_name, machine, client_info
  - row_wait_obj#, blocking_session
- Go back later to get
  - Sql_text from v$sql
  - SQL stats from v$sqlarea
  - Execution plan from v$sql_plan
  - Object info from dba_objects

# Active Session History

- V$ACTIVE_SESSION_HISTORY
  - Data warehouse for session statistics
  - Oracle 10g and higher
  - Data is sampled every second
  - Holds at least one hour of history
  - Never bigger than:
    - 2% of SGA_TARGET
    - 5% of SHARED_POOL (if automatic sga sizing is turned off)
- WRH$_ACTIVE_SESSION_HISTORY
  - Above table gets flushed to this table

# Top Wait Time (52 Customers)

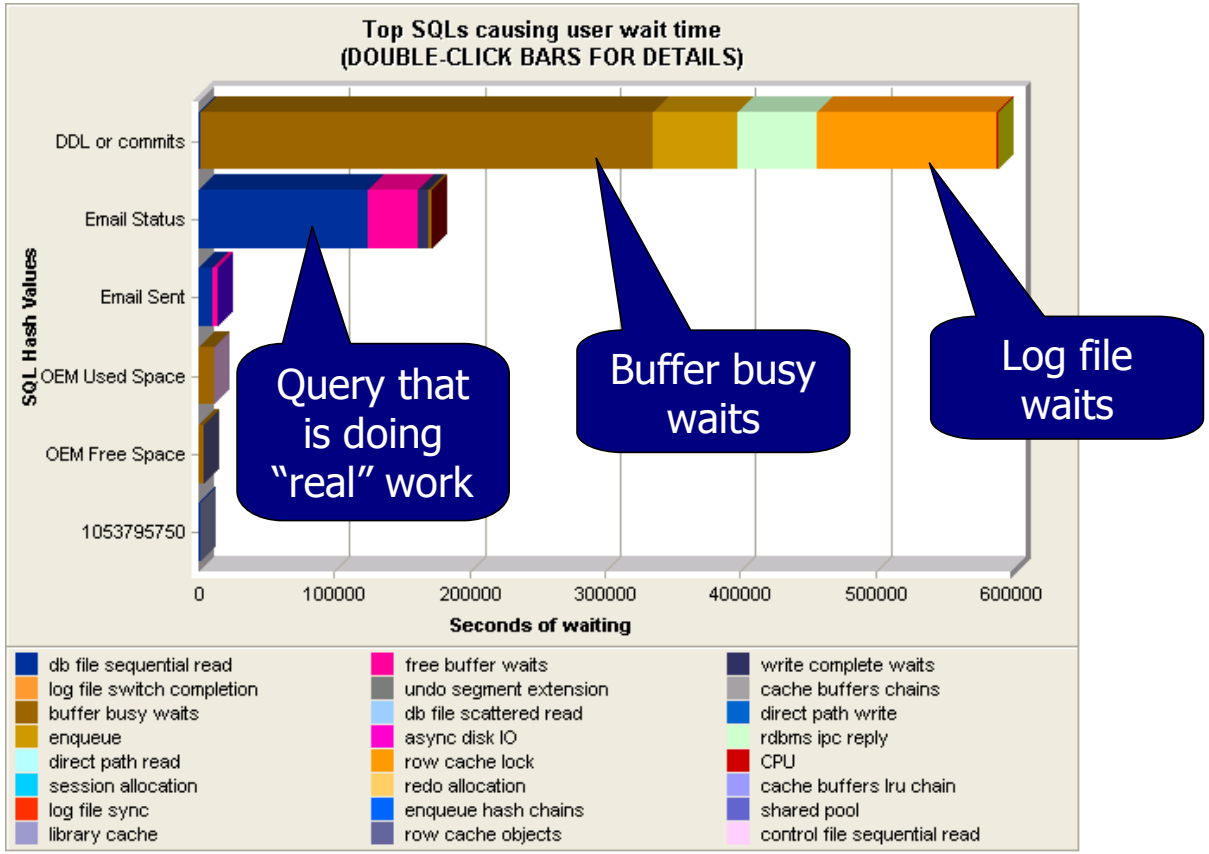
- db file sequential read - 28%
- db file scattered read - 27%
- CPU - 12%
- direct path read / write - 11%
- buffer busy waits - 5%
- log file sync - 3%
- library cache lock - 2%
- log buffer space - 2%

# Case Study One
# Hot Block Issue

# Problem Observed

- Critical situation: application performance unsatisfactory
  - All email coming into and going out of the company was tracked in order to find:
    - Viruses
    - Espionage
    - Also for legal compliance reasons
  - However, email was getting behind
  - Email not getting to end-users for several hours
  - Declared a top priority in company

# Wait Events During Problem



# What do we know?

- Which SQL: DDL or Commits
  SQL hash_value=0
- Which Resource: buffer busy waits
  log file waits
- How much time: 163 Hours of wait time per day

# “buffer busy waits” Description

- Buffer is being read into cache by another session and this session is waiting for that process to complete.
  - In Oracle 10g buffer busy waits are further refined and this becomes “read by other session”
- Buffer is already in the cache but in an incompatible mode, e.g., another session is changing it.

# “buffer busy waits” Description

- P1 – file number information
- P2 – block number information

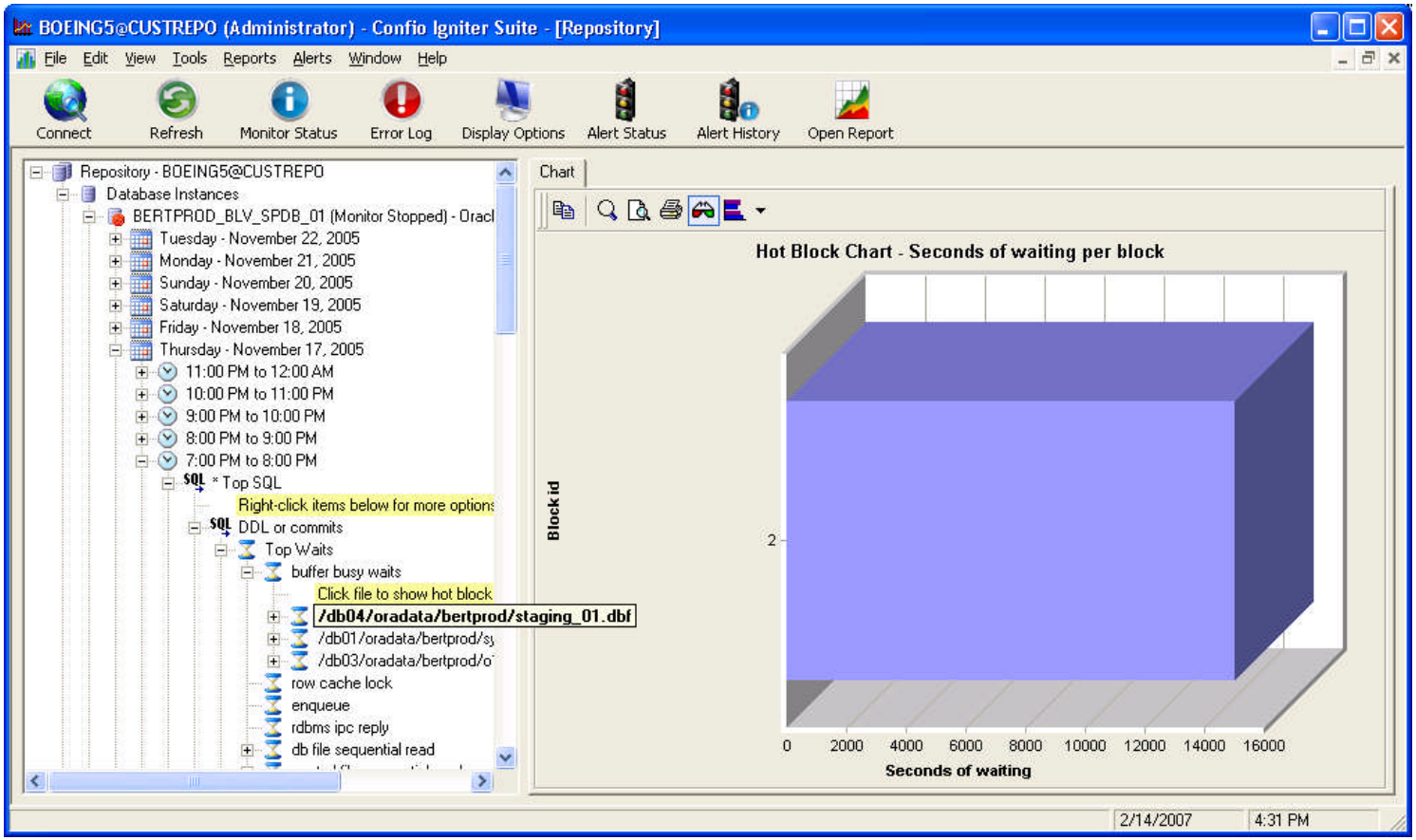
```
SELECT owner, segment_name, segment_type
FROM   dba_extents
WHERE  file_id = &P1
AND &P2 BETWEEN block_id AND block_id + blocks -1
```

- Gives information about the object being waited for

# “buffer busy waits” Description

- Waiting on Data Blocks
  - Tune Inefficient Queries
  - Eliminate Hot Blocks
- Waiting on Segment Headers
  - Optimize PCTFREE / PCTUSED
  - Use Multiple Freelists
  - Use Larger Extents
  - Optimize Application

# “buffer busy waits” Analysis



# Results

- Found hot block problem
  - "buffer busy waits" was waiting for **Block #2** in the file "…staging01.dbf"
  - The email processing code was creating a series of staging tables, every time it executed
- Solutions
  - Started using temporary tables vs. create/drop distinct tables each time the process ran

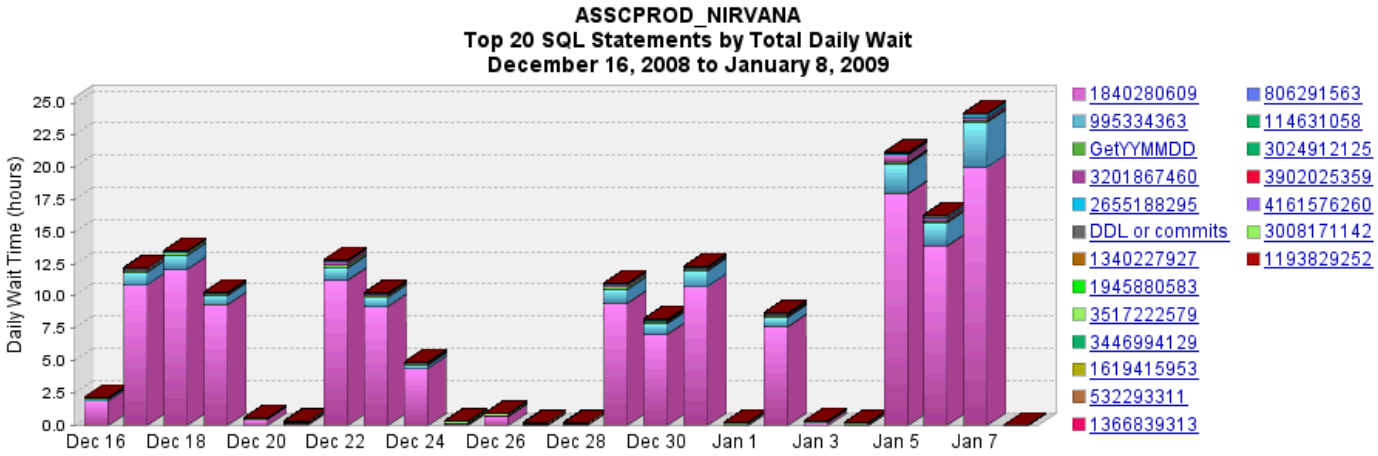
# Case Study Two
# DB File Scattered Reads

# Problem Observed

- Problem: One database using excessive CPU resources on a shared server, impacting other customers
  - High wait accumulation during business hours
  - 20 Hours Every Day
  - Other applications databases were being starved of CPU and I/O resources

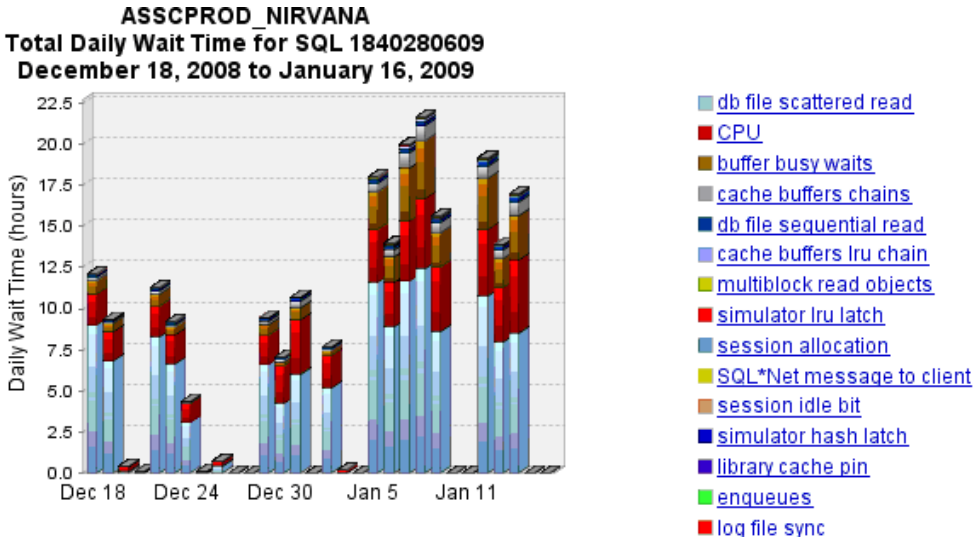
# Investigation

Two statements responsible for nearly all of the wait time



# Investigation

- Lots of wait time for db file scattered reads for SQL 1840280609 (primary consumer from previous slide)



# What do we know?

- Which SQL: 1840280609 (voting tally)
- Which Resource: scattered read buffer busy waits
- How much time: 20+ Hours Every Day

# Hypotheses: Oracle Interpretations

Key Questions:

1. Is full table scan necessary?
2. What causes a full table scan for this SQL Statement?

Two Alternative paths for optimization:

I. Eliminate Full Table Scan
   1. Add Index / Collect Histograms
   2. Update Statistics
   3. Utilize Query Hints

II. Full Table Scan Required - Improve response time
   1. Parallelized Reads
   2. Optimize I/O Subsystem
   3. Optimize Application

# I. Unnecessary Full Table Scan?

Solutions:

1. Add / Modify index(es) on the table
2. Update table and/or index statistics if proper index not being used
3. Add hint to use existing index

# Full Table Scan is Needed

Two alternative paths for optimization:

I. Eliminate Full Table Scan
- There isn't a need to read the whole table, so we need to find the right shortcut

II. Improve response time
- We need to read most or all of the table anyway, so let's just figure out how to do it faster

# II. Improve Response Time for Db File Scattered Reads

Solutions:

1. Use Parallel Reads
2. Set Database Parameters
3. Improve I/O Speed
4. Optimize the application

# 1. Use Parallel Reads = Faster FTS

- Parallel Reads
  - Can be set at the table level (use with caution)

```
Alter table customer parallel degree 4;
```

  - Normally used by hinting in the SQL Statement

```
select /*+ FULL(customer) PARALLEL(customer, 4) */ customer_name
    from  customer;
```

- A delicate tradeoff
  - sacrifice the performance of others for the running query.
  - Parallel Query is designed to maximally utilize your hardware – does not play well with others.
- Not necessarily efficient
  - Just may be faster.
  - Parallel Reads may actually do twice the work of a serial query but have four workers, thus finishing in half the time while using 8x resources

# 2. Set database parameters

- DB_FILE_MULTIBLOCK_READ_COUNT
  - specifies the maximum number of blocks read in one I/O operation during a sequential scan
  - Impacts the optimizer
  - Reduces number of system I/Os calls required to read a set of data
  - For OLTP, typically set to between 4 to 16, in newer versions of Oracle it may be best left unset
  - Optimizer will more likely to FTS if set too high
- Ensure that the database read requests are synced up with the O/S.
- This gets tricky if different block sizes are used in different tablespaces

# 3. Improve I/O speed

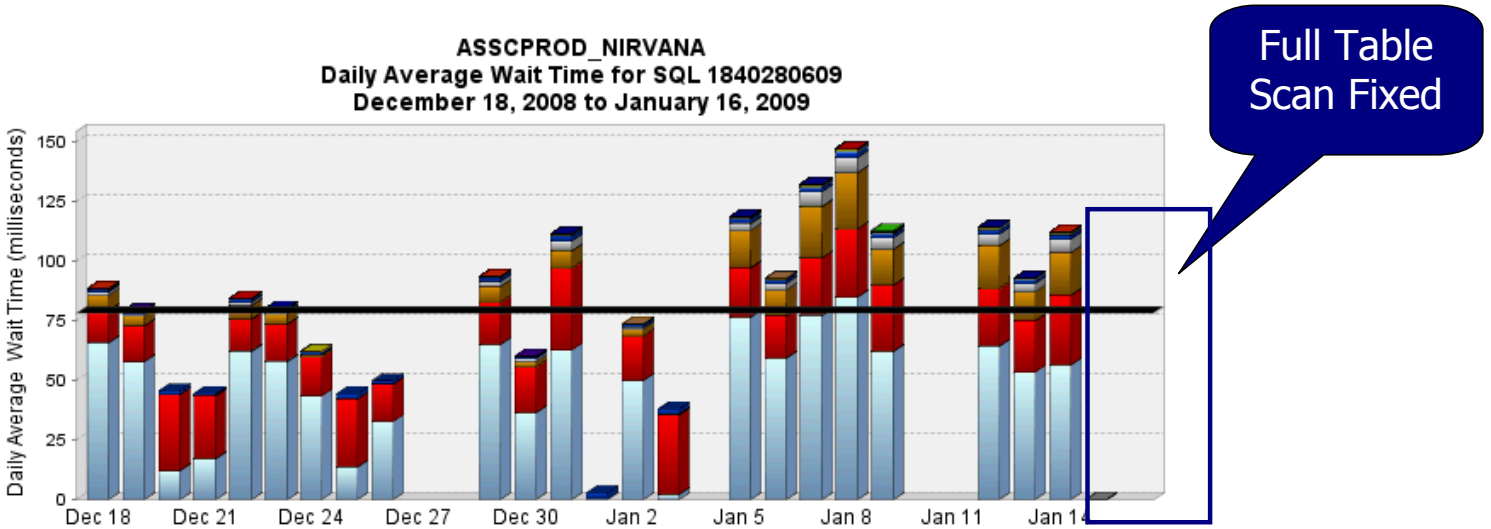
- Get your SA involved
- Investigate I/O sub-system
  - Iostat, vmstat, sar, … for potential problems
  - Monitor during high activity
  - There may be a whole host of things wrong
- Investigate contention at the disk/controller level.
  - Learn which disks share common resources
  - Use more disks to spread I/O and reduce hot spots
- Investigate caching and current memory usage

# 4. Optimizing the Application

- Review application – do you have access to code for changes?
  - No – look into stored outlines
- Techniques to Optimize a statement:
  - Reduce the number of calls for a SQL
    - Caching the data in the application
    - Creating a summary table (perhaps via a materialized view)
    - Eliminating the need for the data
  - Retrieve Less Data with each statement
    - Add fields to the WHERE clause

# Results

- Added indexes to table



# Results

- Statement statistics – 24 hour snapshots

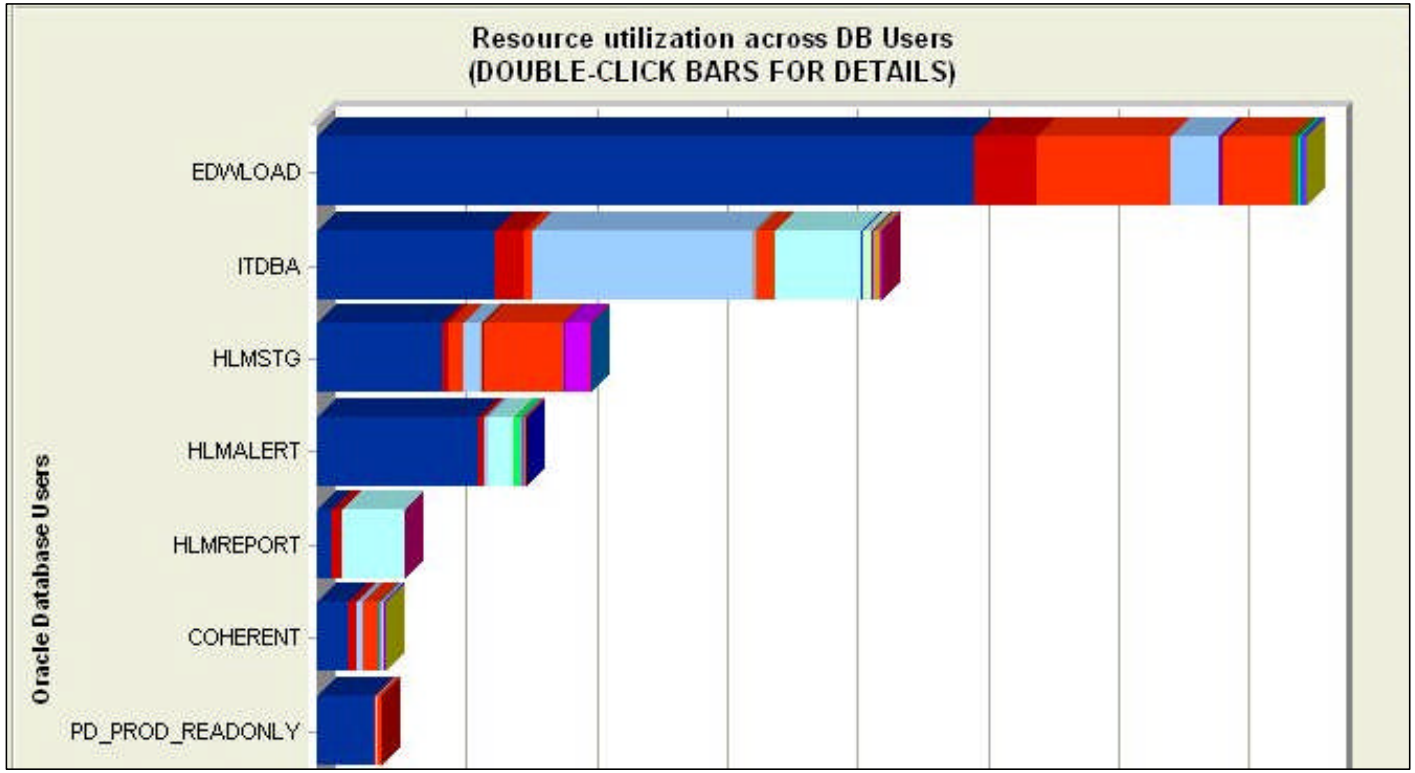
| Statistic | Before | After | Difference |
|---|---|---|---|
| Executions | 554,872 | 539,147 | (15,725) |
| Rows processed | 554,870 | 539,170 | (15,700) |
| Parses | 502 | 568 | 66 |
| Disk reads | 766,490,774 | 0 | (766,490,774) |
| Buffer gets | 1,300,242,461 | 2,064,536 | (1,298,177,925) |
| | | | |

# Case Study Three
# DB File Sequential Reads

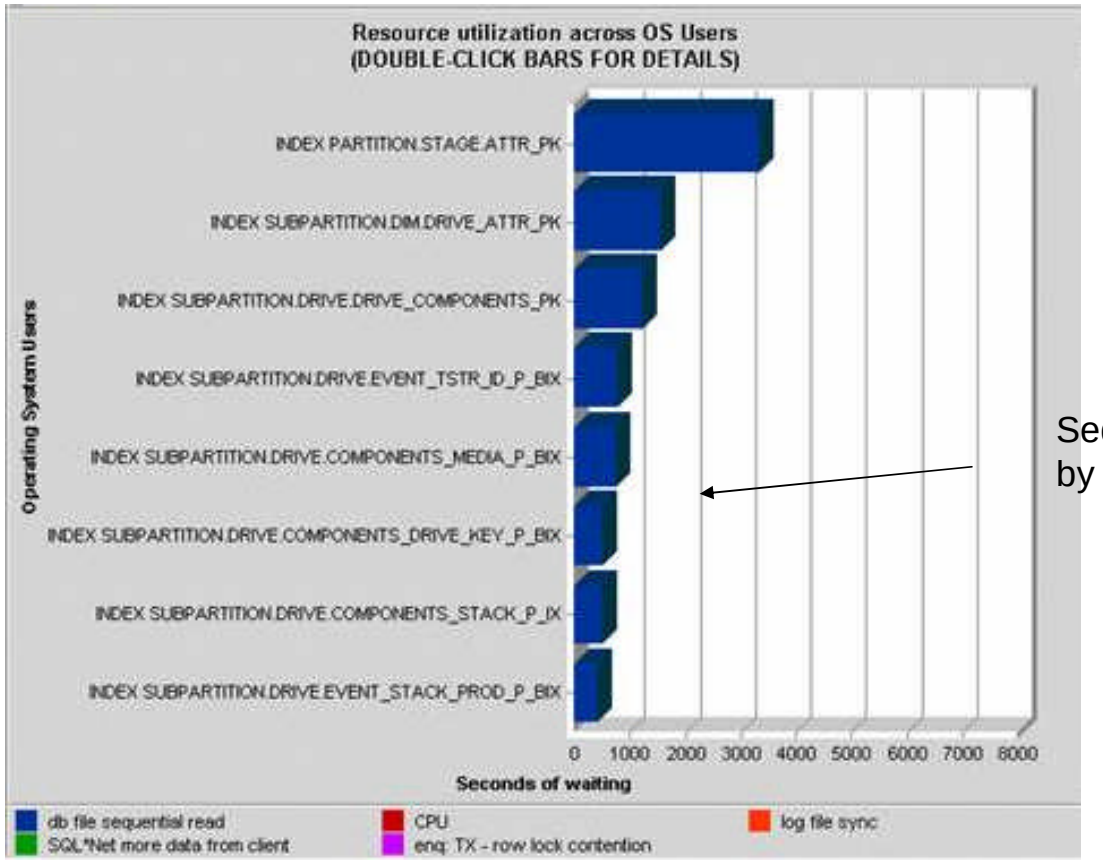
# Problem Observed

- Data Warehouse loads were taking too long
- Noticed high wait times on "db file sequential read" wait event
- DBAs were confused – why are data loads "reading" data

# Investigation



# Investigation for an INSERT Statement




Sequential read time by object for SQL

# What do we know?

- Which SQL: Load Process
- Which Resource: DB File Sequential Read
- How much time: 5 hour+
  90% of wait time

# Investigating db file sequential reads

- Often considered a “good” read
- DB file sequential reads normally occur during index lookups
- Single-block read
  - P1 – file id
  - P2 – block id
  - P3 – number of blocks read
  - Join to DBA_EXTENTS (see buffer busy waits)

# Hypotheses: Oracle Interpretations of Sequential Reads

Causes of excessive wait times:

I. Reading too many index leaf blocks
II. Low cardinality first column index
III. Not finding block in buffer cache forces disk read
IV. Slow disk reads

# I. Reading too many index and table blocks (cont)

1. Rebuild Fragmented Indexes
   - alter index rebuild [online];
   - Consider http://www.jlcomp.demon.co.uk/index_efficiency.html
2. Compress Indexes
   - alter index rebuild compress;
   - Uses more CPU
3. Multi-column indexes
   - Avoid the table lookup
   - Will create a larger index
4. Pre-sort Table data

# II. Low cardinality first column index

- If first column of index is low / medium cardinality, much time is spent scanning the index leaf blocks
- The additional columns do not lower the number of leaf blocks that are read.
- Solutions:
  - Use a leading column with better cardinality
  - Compress the index

# III. Not finding block in buffer cache forces disk read

- Db File sequential reads occur because the block is not in the buffer cache.
- How do we make sure more blocks are already in the cache?
- Solutions
  1. Increase the size of the buffer cache(s)
  2. Put the object in a cache where it is less likely to get flushed out

# IV. Slow disk reads

- With databases, it often comes down to this – the disk just needs to be faster
- Put certain objects on the fastest disk
- O/S file caching using special software that makes normal files perform like raw files
- Increase Storage System Caching – such as an EMC cache

# Results

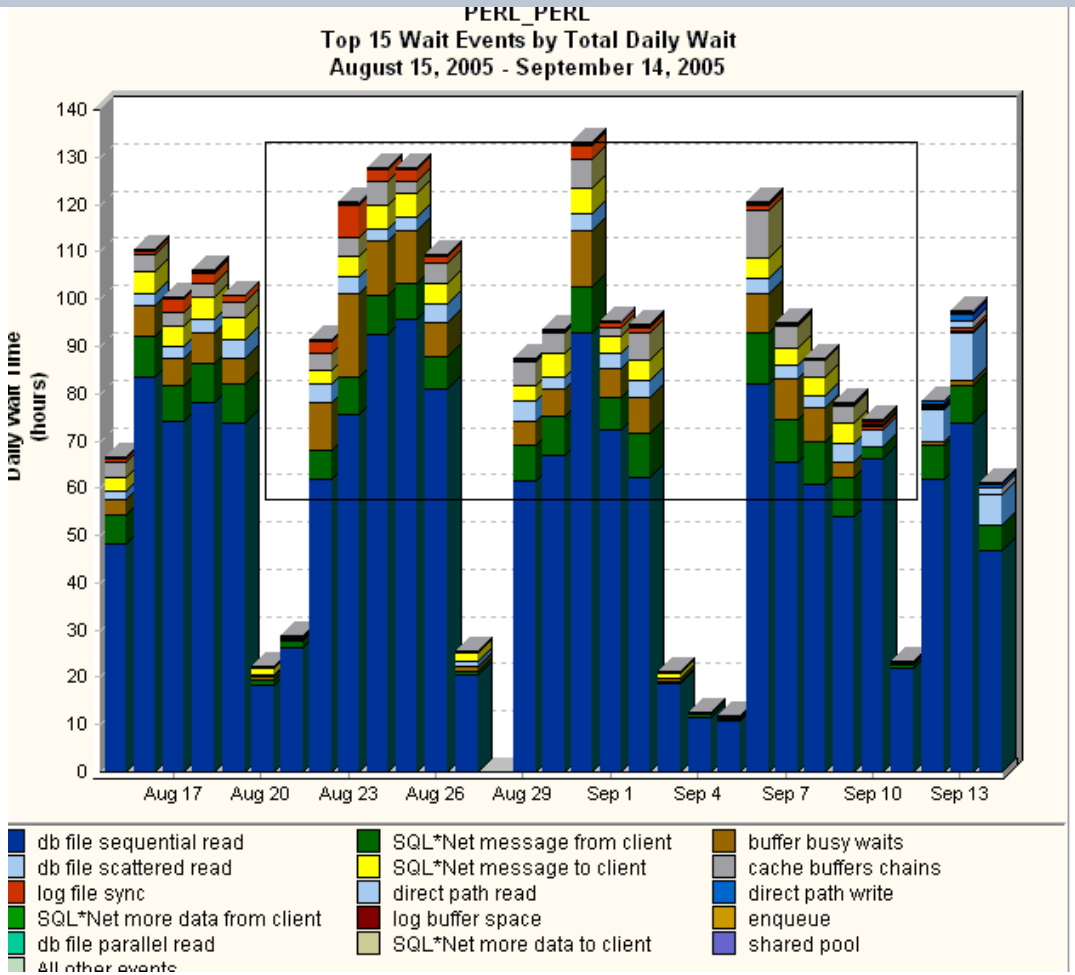
- Many sessions were loading data and all were updating low cardinality indexes
- Modified index and noticed a 50% performance improvement in an INSERT
- Customer is also analyzing global vs. local indexes
- Reviewing usage of bitmap indexes
- Removed unused indexes
- Enhanced the disk subsystem

# Case Study Four
# Cache Buffers Chains Latching

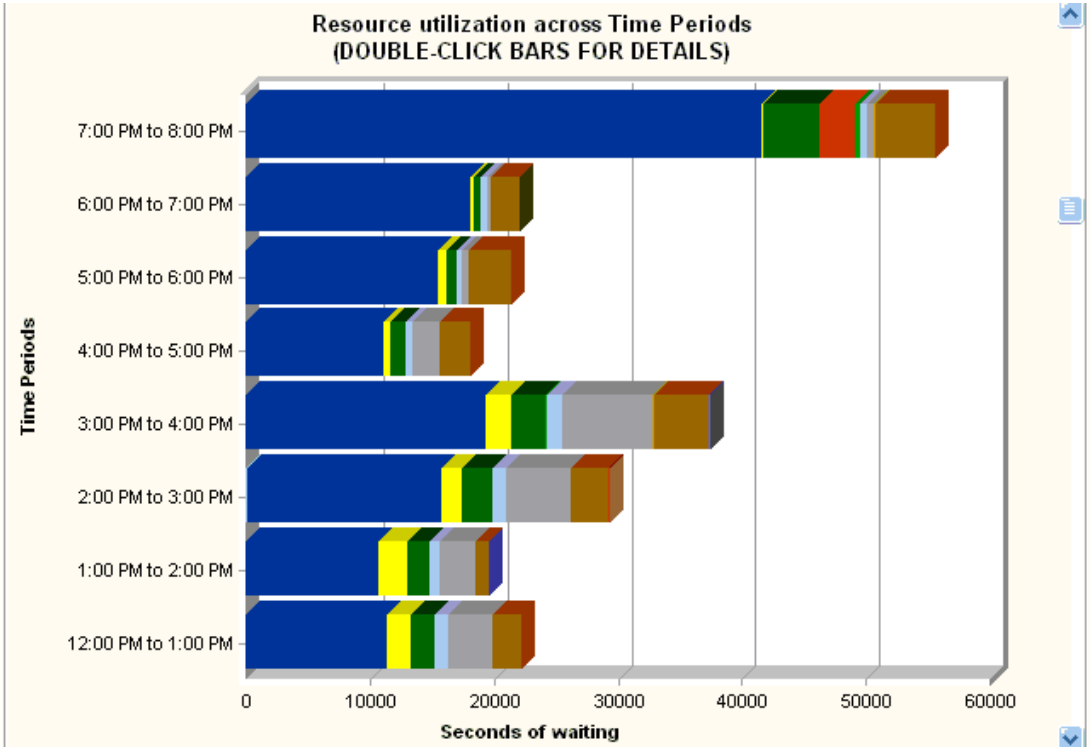
# Problem Observed

- Problem: High Wait on Production CRM Database
  - Accumulated wait 20 hours (72,000 sec) during peak online hours
  - End users in Virginia were complaining vigorously – they could not perform their jobs

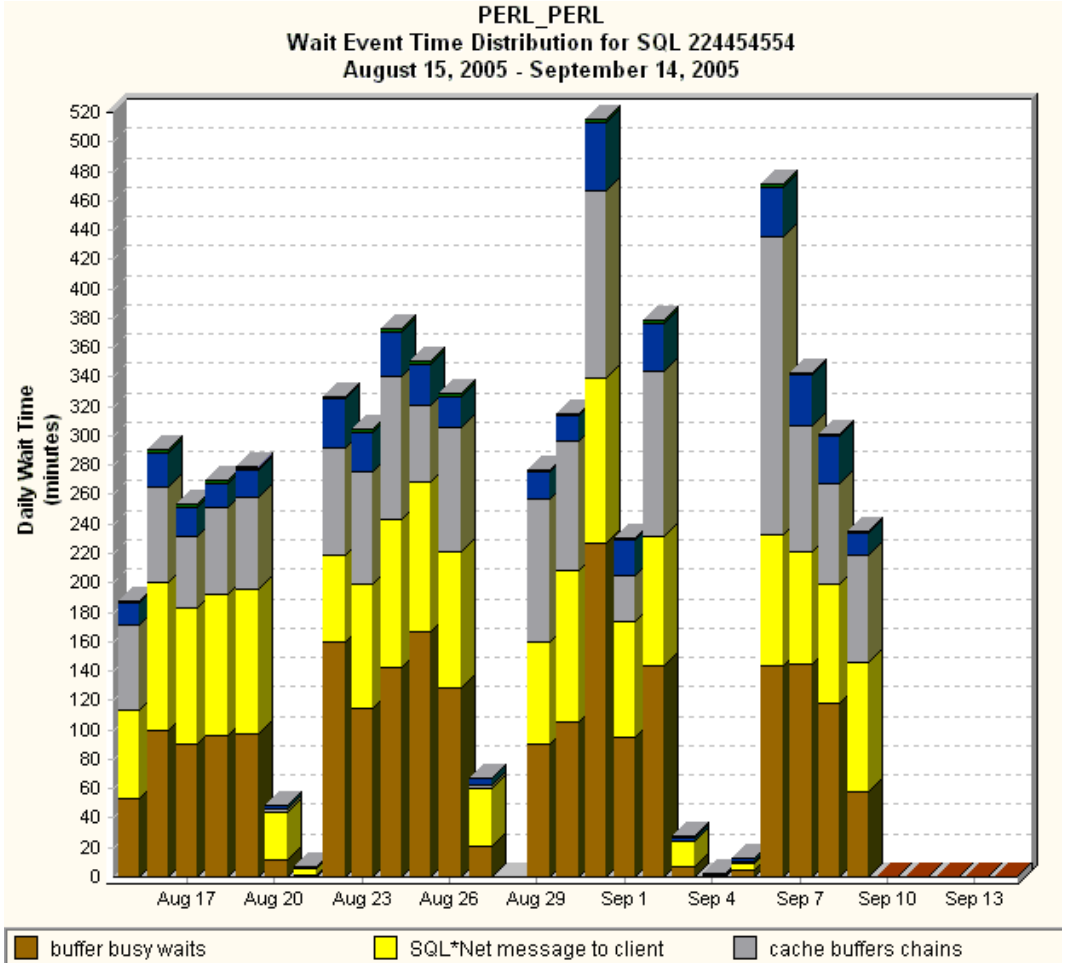
# Investigation



# Investigation – which events?



# Investigation – SQL statement



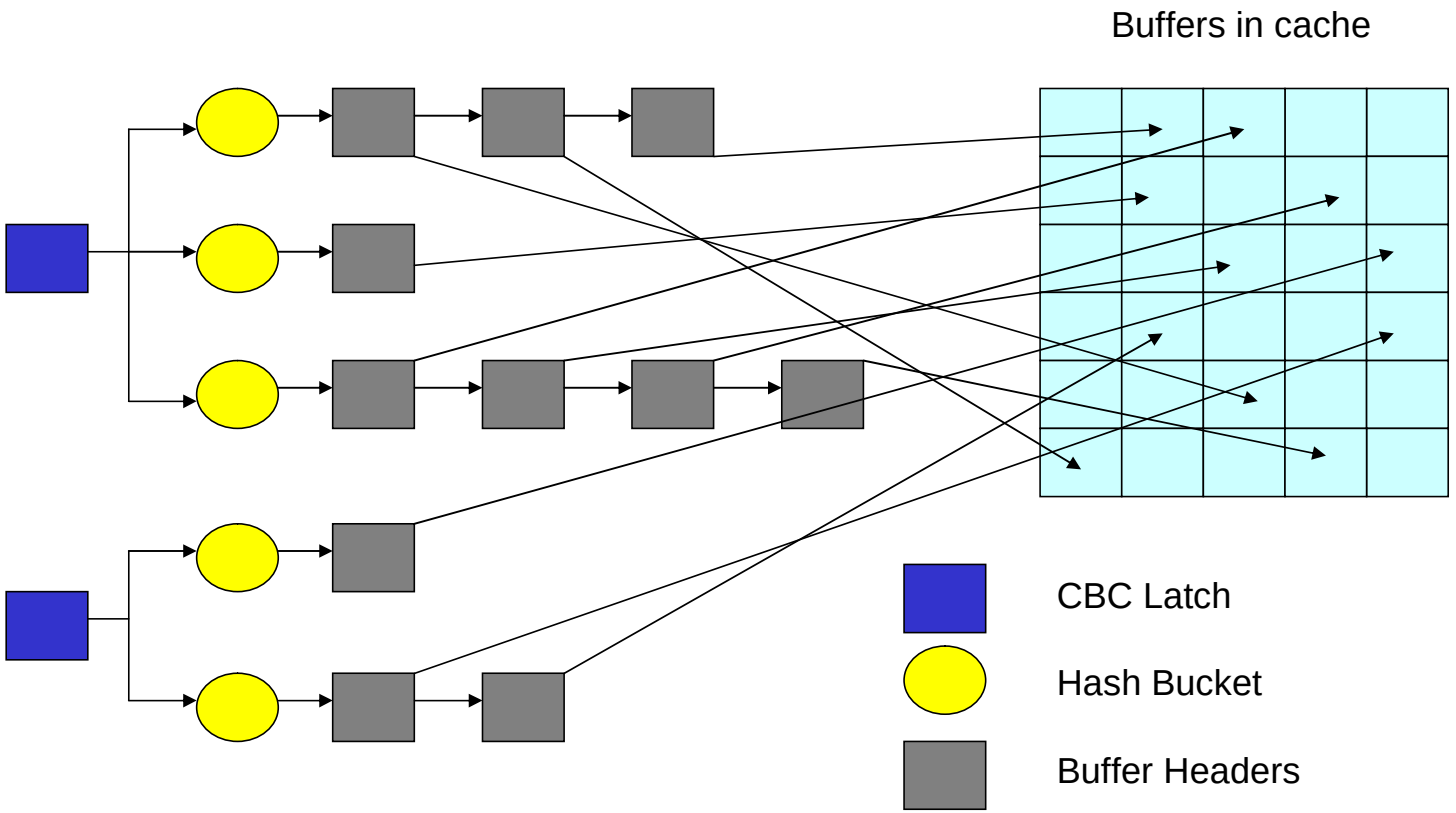
# What do we know?

- Which SQL: 224454554
- Which Resources: CBC latching<br>buffer busy waits
- How much time: A bunch<br>every weekday

# Cache Buffers Chains Latching

- CBC latches protect memory structures called hash chains, which hang off hash buckets
- The hash chains are fixed-depth linked lists of buffer headers that point to the actual buffers in the cache
- The data block address (DBA) is hashed to find the hash chain to be searched to find the location in cache
- Accessing a block on the hash chain requires acquisition of the latch protecting that particular chain (9i and later allows shared read-only access for some operations – although apparently not FTS)
- This prevents other processes from modifying the chain

# CBC Latches and Hash chains



# Cache Buffers Chains Latching

- Excessive CBC latching can be problematic because
  - All LIOs require latch gets
  - CBC latches cover many buffer headers, and only one process can exclusively hold a CBC latch
  - Once a process goes to acquire a latch, it either gets the latch or spins/sleeps until it gets the latch
  - Latching is not FIFO
  - Latching drives up CPU

# Cache Buffers Chains Latching

- Causes
  - Inefficient SQL statements
    - Too many LIOs
    - High concurrency
    - Bad plans
  - Hot Blocks
    - Find hot blocks via v$session_wait.p1raw (latch address)
    - Confirm hot blocks by looking at touch count in x$bh.tch for x$bh.hladdr (from v$session_wait)
    - Map x$bh.obj to object_id or data_object_id from dba_objects
    - Determine why the application hits those blocks so frequently
  - Long hash chains
  - A combination of the first two

# Finding chains with hot blocks

```
select sid, p1raw, p2, p3, seconds_in_wait, wait_time, state
from v$session_wait
where event = 'latch free'
order by p2, p1raw;
```

p1raw is the latch address
p2 is the latch number
Sessions waiting on the same latch address shows you have hot blocks on a chain

From OWI book.

# Cache Buffers Chains Latching

- Cures
  - Find and fix culprits
    - Tune SQL to require fewer LIOs
  - Reduce concurrency
    - Usually by fixing the application

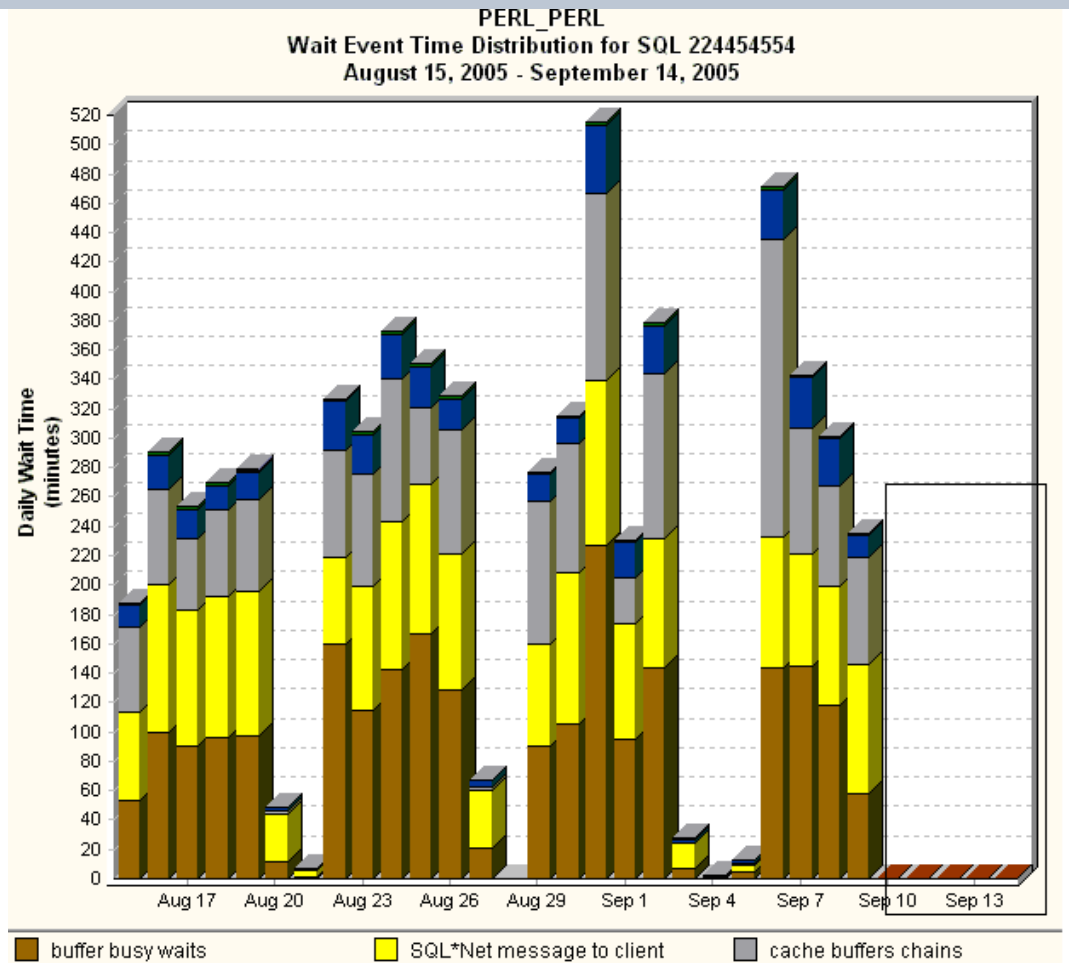
# In this case…

- Inefficient SQL, made worse by high concurrency
- SQL was not using indexes, lots of LIOs
- Lots of folks running the SQL
  - It took too long
  - The application must be hung! I'll resubmit…
    - Again
    - And again

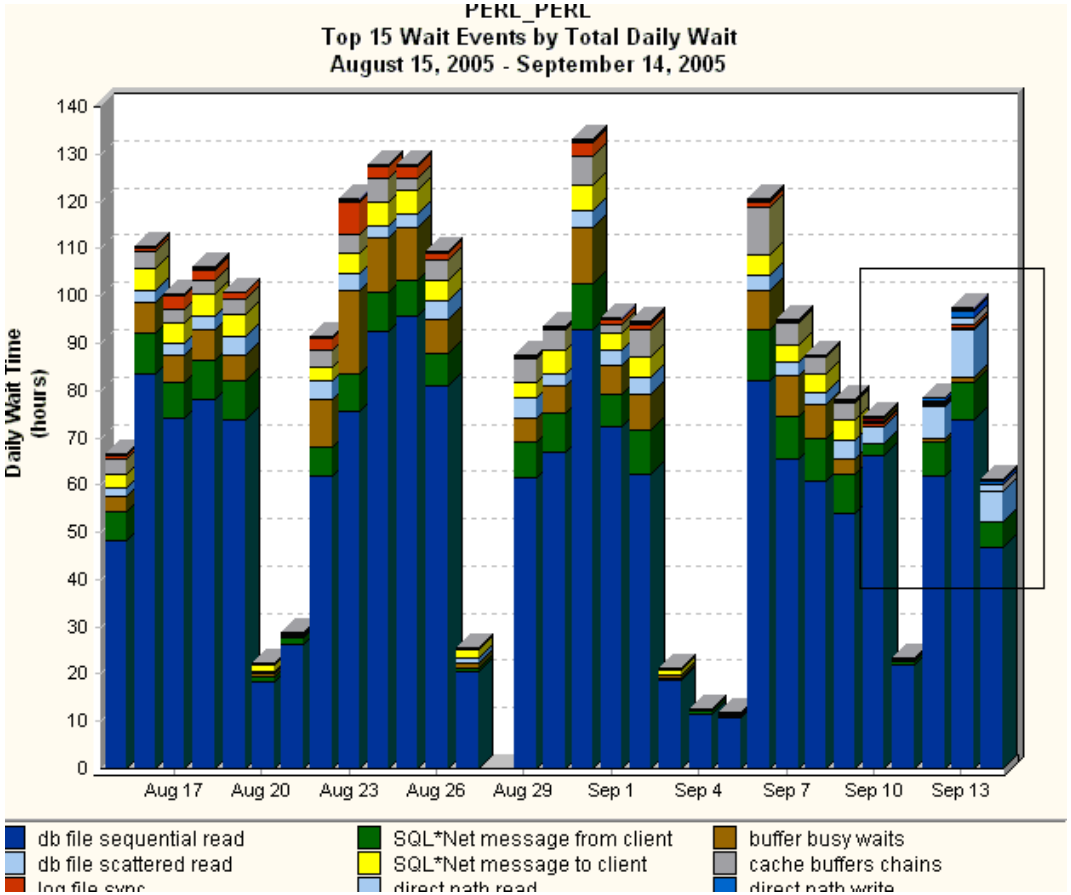
# Results

- SQL statement was examined, and determined that an implicit type conversion was causing indexes to not be used: comparing numbers and character strings
- Showed up in the filter predicates from dbms_xplan output
- Changed the code to use the correct values

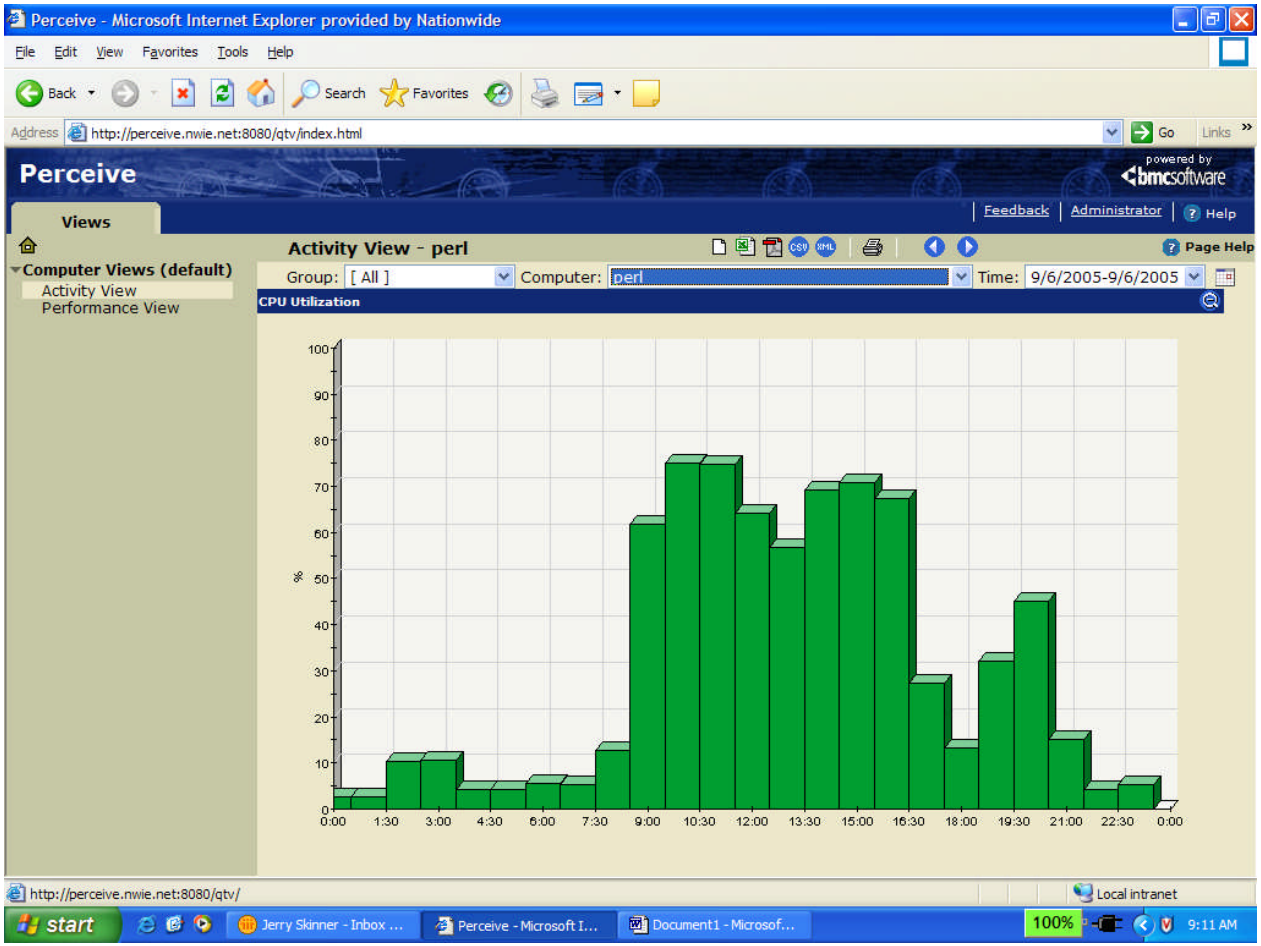
# Results


PERL_PERL
Wait Event Time Distribution for SQL 224454554
August 15, 2005 - September 14, 2005
Daily Wait Time (minutes)
520
500
480
460
440
420
400
380
360
340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Aug 17
Aug 20
Aug 23
Aug 26
Aug 29
Sep 1
Sep 4
Sep 7
Sep 10
Sep 13
buffer busy waits
SQL*Net message to client
cache buffers chains

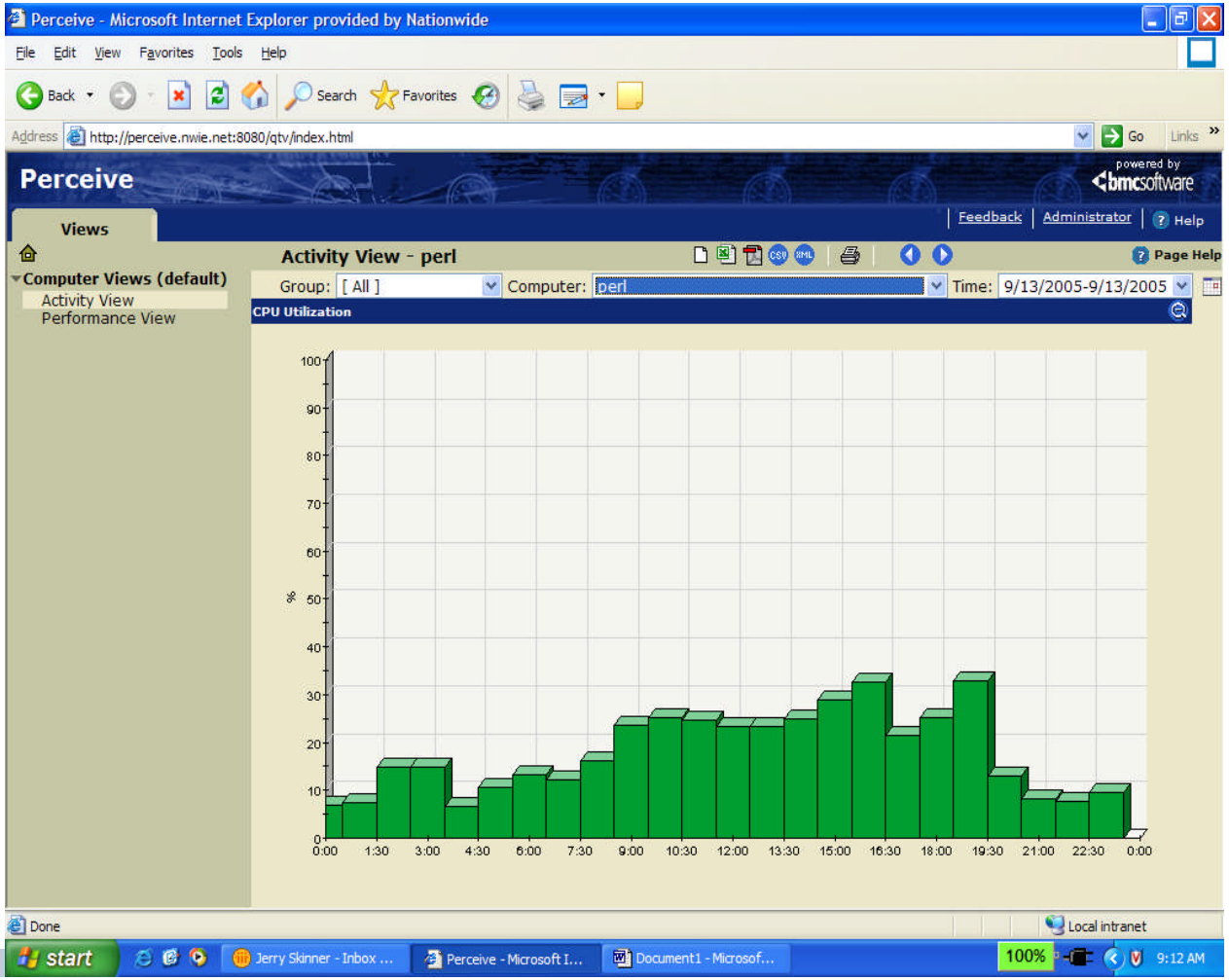
# Results


PERL_PERL
Top 15 Wait Events by Total Daily Wait
August 15, 2005 - September 14, 2005
Daily Wait Time (hours)
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
Aug 17
Aug 20
Aug 23
Aug 26
Aug 29
Sep 1
Sep 4
Sep 7
Sep 10
Sep 13
db file sequential read
db file scattered read
log file sync
SQL*Net message from client
SQL*Net message to client
direct path read
buffer busy waits
cache buffers chains
direct path write

# CPU consumption (before)



# CPU consumption (after)



# Wait Events for Development

- Tuning SQL for optimal performance
- Debug/test/integrate/pilot process
- Understand impact on existing database
- Understand Oracle impact on application performance
- View into production for better development prioritization and feedback
- Reduce finger-pointing

# Conclusion

- Conventional Tuning focused on "system health" and lead to finger-pointing and confusion
- Wait event tuning implemented according to three principles is the best way to tune
- Two compliant tools types
  - Trace profiling tools
  - Continuous DB-wide monitoring tools

# References

Shee, R.; Deshpande, K.; Gopalakrishnan, K. 2004. *Oracle Wait Interface: A Practical Guide to Performance Diagnostics & Tuning*, McGraw-Hill/Osborne

# Questions?

# Thank you!

Thank you for coming to this presentation

Contact information for further questions

- deedsd@nationwide.com
  - Please put "wait time presentation in the subject.