ORACLE®

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE OPEN WORLD

Your. Open. World.

Real-World Database Performance Techniques and Methods

Andrew Holdsworth
Senior Director Real-World Performance, Server Technologies

ORACLE

Real-World Performance Schedule 2008

Real World Performance 2008

Session ID: S299785
Session Title: Growing Green Databases with Oracle on the UltraSPARC CMT Processor
Track: Database
Venue: Moscone South
Room: Rm 236
Date: 2008-09-22
Start Time: 13:00

Session ID: S298786
Session Title: Current Trends in Real-World Database Performance
Track: Database
Venue: Moscone South
Room: Rm 103
Date: 2008-09-23
Start Time: 13:00

Session ID: S298792
Session Title: Real-World Database Performance Techniques and Methods
Track: Database
Venue: Moscone South
Room: Rm 104
Date: 2008-09-25
Start Time: 12:00

Session ID: S298785
Session Title: Real-World Database Performance Roundtable
Track: Database
Venue: Moscone South
Room: Rm 104
Date: 2008-09-25
Start Time: 13:30

Real-World Database Performance Techniques and Methods

Agenda

- Optimizer Exposé
- Managing Statistics on Partitioned Tables
- When to Apply the Knife to your Data/Workloads/ Databases
- Detecting and Avoiding Hiccups in your OLTP System

Optimizer Exposé

Optimizer Exposé

Common Feedback

- The “Optimizer” has a personality and moods all of its own. Some statements have different plans at different times of the day.
- The “Optimizer” never seems to use the correct index.
- The “Optimizer” scans table when I want to use index access and uses an index when I want a table scan.
- Why are nested loops so bad sometimes?
- It is impossible to figure out what the Optimizer is doing, and why?
- What statistics should I build to get good execution plans? How do I do it?
- There are hundreds of pages of Optimizer related documentation; where do we start?

Optimizer Exposé

Upgrading Issues

- The upgrade to Oracle 10g has proven difficult because of changes in the default behavior of the Optimizer and **`DBMS_STATS`**. There is a real lack of understanding of the impact of these changes.
- This has impacted production systems in terms of poor performance when execution plans degrade.
- Unpredictable performance when execution plans change in an unpredictable manner.
- No formal debugging methodology.
- The tendency to make global configuration changes to fix a small number of SQL statements.

Optimizer Exposé

Upgrading Issues

- The move to automatic statistics gathering has resulted in the following issues:
  - Execution plans can change literally overnight due to new statistics
  - New histograms may be created
  - Bind peeking becomes an issue because of new histograms
- Unfortunately most sites don’t know their execution plans prior to upgrade
- Without a formal debugging method, anarchy prevails with continual “hacking” of init.ora parameters and schema statistics to fix degraded SQL statements

Optimizer Exposé

Plan Predictability vs. Plan Evolution

- A common request is “Can you make the Cost Based Optimizer more like the old Rule Based Optimizer?”
- The way the DBA chooses to create schema statistics will have huge impact on the challenge of Plan Predictability vs. Plan Evolution.
- To make the Optimizer more predicable, the best way is to restrict the variable or non-predictable components of its functionality.
- This may mean you don’t get the optimal execution plan, but at least you get the same one each time, which may be the preferred behavior.

Optimizer Exposé

Plan Predictability vs. Plan Evolution

- Ways to achieve consistency of Execution Plans
  1. Design statistics gathering strategy to deliberately exclude histograms and police column high/low values very aggressively.
     - This will assume uniform distribution of data because no histograms exist.
     - Accurate high/low values are crucial to prevent out of range cardinality estimates.
  2. Use Oracle Tools (SQL Profiles, Outlines, etc.) These tools use hindsight to optimize the SQL statements.
  3. Manually hint every SQL statement (effectively removing the Optimizer from the problem !)

Optimizer Exposé

Plan Predictability vs. Plan Evolution

- These approaches will not guarantee the best execution plans but should result in reliable and predictable plans.
- Any non-performing plans can be manually corrected.
- This approach is very suited to OLTP type applications where consistency is very important.

Optimizer Exposé

Plan Predictability vs. Plan Evolution

- Allowing execution plans to evolve
  - This will involve many iterations of statistics gathering.
  - You will need to re-test your application for execution plan issues when gathering statistics, or be willing to accept the occasional poor plan in production.
  - You should understand how to debug poorly optimized SQL and recognize the root cause of problems.
  - If your approach is to hack init.ora parameters to resolve execution plan issues, you probably should not be doing this as you are clearly out of your depth. There are too many variables involved to manage effectively.
- The evolutionary approach is well suited to DW/BI databases where the optimizer often has only one chance to get it right.

Optimizer Exposé

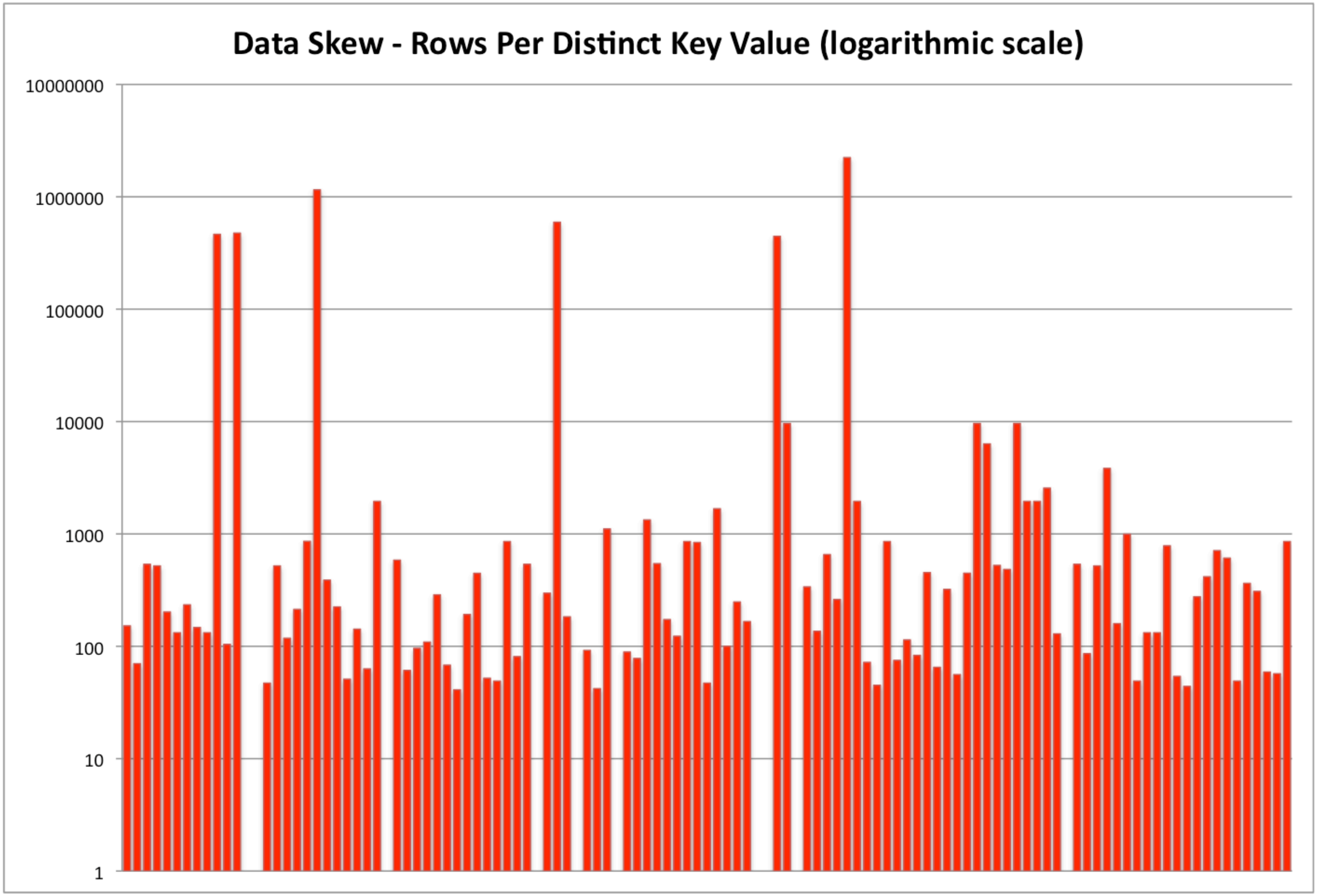
The Six Challenges to the Cost Based Optimizer

- The six most common challenges are related to:
  1. Data skew
  2. Bind peeking
  3. Column low/high values
  4. Data correlation between columns
  5. Cardinality Approximations
  6. The debugging process
- These challenges are all interrelated and the solution to solve one challenge may trigger another challenge.
- Working on these issues is complex and frustrating.
- In the end pragmatism will win over idealism.

Optimizer Exposé

Challenge #1: Data Skew

- Definition:
  - a non-uniform distribution of data, generally on a per column, per value basis
- Diagnose by:
  - Simple query with GROUP BY to show skew
  - Poor cardinality estimates
  - Estimates may be wrong by orders of magnitude
- Potential Solutions:
  - Determine if uniform plans or variable plans are desired
  - Histograms (be aware of bind peeking)
  - Both global and partition level stats


Data Skew - Rows Per Distinct Key Value (logarithmic scale)
10000000
1000000
100000
10000
1000
100
10
1

Optimizer Exposé

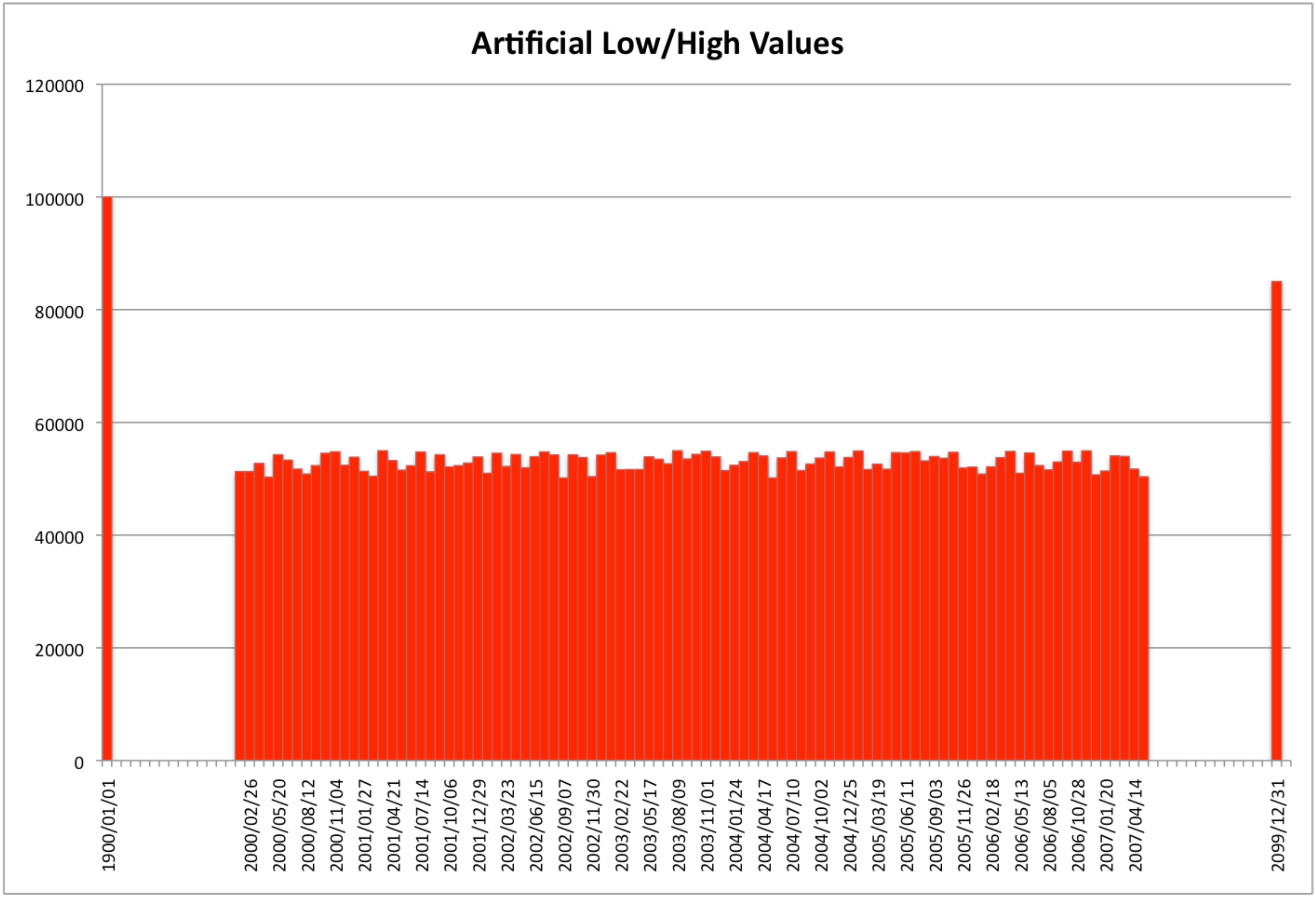
Challenge #2: Bind Peeking

- Definition:
  - The query optimizer peeks at the values of user-defined bind variables on the first invocation of a cursor. This feature enables the optimizer to determine the selectivity of any WHERE clause condition as if literals have been used instead of bind variables.
  - Histograms, high values, single partition access can impact plan choice
- Diagnose by:
  - Execution plans change due to cursors aging – flip flop plans
  - Plans differ across cluster instances
- Potential Solutions:
  - To insure consistency of plans, remove/manage statistics that lead to variable execution plans
    - Eliminate histograms, or use literal predicates for columns with histograms if alternate plans are required
    - Manage column low/high values so binds can not be out-of-range
    - Use global statistics so no single partition optimization takes place

Optimizer Exposé

Challenge #3 Column Low/High Values

- Definition:
  - `DBMS_STATS` gathers the low and high value for each column
- Diagnose by:
  - Stale statistics may cause out-of-range predicates to have underestimated cardinality. Be very suspicious of cardinality estimates of 1
- Potential Solutions:
  - Regather stats
  - Manually set statistics to adjust the high value

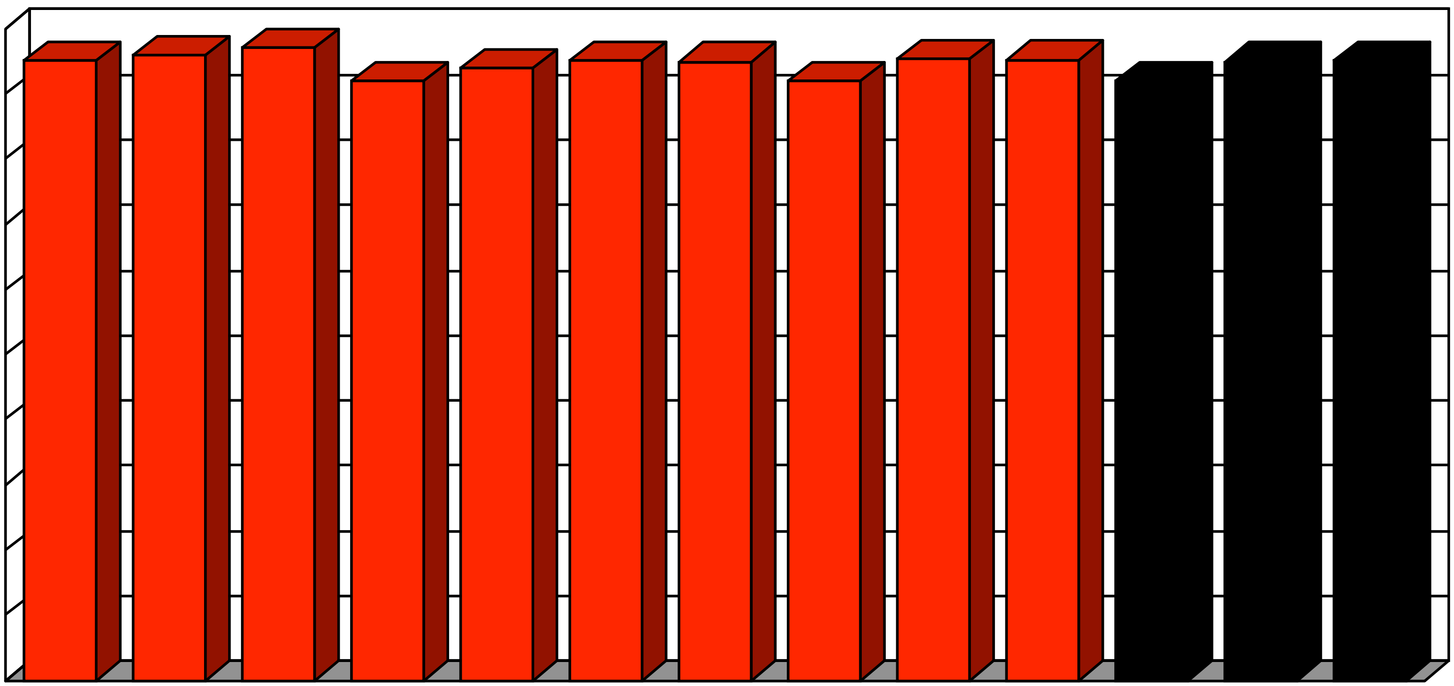

Artificial Low/High Values
120000
100000
80000
60000
40000
20000
0
1900/01/01
2000/02/26
2000/05/20
2000/08/12
2000/11/04
2001/01/27
2001/04/21
2001/07/14
2001/10/06
2001/12/29
2002/03/23
2002/06/15
2002/09/07
2002/11/30
2003/02/22
2003/05/17
2003/08/09
2003/11/01
2004/01/24
2004/04/17
2004/07/10
2004/10/02
2004/12/25
2005/03/19
2005/06/11
2005/09/03
2005/11/26
2006/02/18
2006/05/13
2006/08/05
2006/10/28
2007/01/20
2007/04/14
2099/12/31

The High/Low Value Cardinality Challenge

Easy to compute # of rows

Difficult to compute the number of rows as [illegible] above the max value

Impos[illegible] of rows because lower an[illegible] upper range above max value



Optimizer Exposé

Challenge #4: Data Correlation Between Columns

- Definition:
  - Two or more columns have values that are related to one another
  - Often times hierarchical data
  - e.g. country, state, region, zip code
- Diagnose by:
  - Cardinality estimates are too small. Look out for Cardinality estimates of 1.
- Potential Solutions:
  - Dynamic sampling
  - Multi-column statistics in 11g

Correlation Example

- Consider a table with 5 columns of number datatype
- For any given row, all 5 columns have the same value
- The range of values is 1 through 10
- Each combination has 10,000 rows
- Table has 100,000 rows

Correlation Example (2)

```
SQL> select c1,c2,c3,c4,c5,count(*) from correlation
     group by c1,c2,c3,c4,c5 order by c1,c2,c3,c4,c5;

  C1        C2      C3      C4      C5    COUNT(*)
------ ------ ------ ------ ------ ----------
     1      1      1      1      1      10000
     2      2      2      2      2      10000
     3      3      3      3      3      10000
     4      4      4      4      4      10000
     5      5      5      5      5      10000
     6      6      6      6      6      10000
     7      7      7      7      7      10000
     8      8      8      8      8      10000
     9      9      9      9      9      10000
    10     10     10     10     10      10000

10 rows selected.
```

Correlation Example (3)

Cardinality =	Cardinality =
number of rows in table *	100,000 *
selectivity predicate1 *	1/10 *
selectivity predicate2 * …	1/10 * …
selectivity predicateN	1/10

Predicate	Estimated Cardinality	Actual Cardinality
C1=1	10,000	10,000
C1=1 and C2=1	1,000	10,000
C1=1 and C2=1 and C3=1	100	10,000
C1=1 and C2=1 and C3=1 and C4=1	10	10,000
C1=1 and C2=1 and C3=1 and C4=1 and C5 =1	1	10,000

Optimizer Exposé

Challenge #5: Cardinality Approximations

- Definition:
  - When using functions the optimizer assumes a cardinality as a percentage of the rows.
  - UPPER, SUBSTR are common
  - Optimizer assumes 1% for equality estimates and 5% for others
- Diagnose by:
  - Cardinality estimates are wrong
- Potential Solutions:
  - Functional indexes
  - Hinting

Optimizer Exposé

Challenge #6 The Debugging Process

- Definition:
  - Bad plans trying to be fixed as a single event
  - Support gives resolution based on one SQL statement
  - > 95% of SQL plans are good, then fix as one offs
  - Manual optimization may be necessary for a few
  - Use stop loss approach
- Diagnose by:
  - Large number of bad plans: probably bad statistics strategy
  - Few number of bad plans: may be edge case
- Potential Solutions:
  - Understand the root cause of the problem and avoid the temptation to hack global changes to fix a single statement

Debugging the Optimizer

Common Feedback

- Where do I start?
- Do I need to be worried about a plan with a high cost?
- How is cost calculated?
- What statistics/values are used in calculating cost?
- Very little debug information
  - 10053 trace impossible to read and doesn't contain everything I can use

Debugging the Optimizer

The common chain of events

- Non representative statistics leads to
- Poor cardinality estimates which leads to
- Poor access path selection which leads to
- Poor join method selection which leads to
- Poor join order selection which leads to
- Poor SQL execution times

Debugging the Optimizer

Systematic Top Down Approach

- Start with cardinality. If the cardinality estimate is bad, the rest of the plan will likely be bad
- If the cardinality estimate for a table is way off, validate the statistics
- Recent statistics <u>are not</u> the same as representative statistics!
- Sanity check **`USER_TAB_COL_STATISTICS.NUM_DISTINCT`** for the columns that have predicate filters. Does the value make sense?
- Beware of row source estimates of 1 if the access path is not via primary key

Debugging the Optimizer

Simple Tool Box

- 11g
  - **DBMS_SQLTUNE.REPORT_SQL_MONITOR**
  - On by default when execution is >5 seconds
  - Can be forced with MONITOR hint
- 10g
  - **GATHER_PLAN_STATISTICS** hint used with **DBMS_XPLAN.DISPLAY_CURSOR(format=>'ALLSTATS LAST')**
- 9i
  - Manual approach to assemble data, concept remains the same: first check cardinality estimates

GATHER_PLAN_STATISTICS and DBMS_XPLAN.DISPLAY_CURSOR

```
select /*+ gather_plan_statistics */ ... from ... ;
select * from
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-------------------------------------------------------------------------------------------
|  Id | Operation                             | Name          | Starts | E-Rows | A-Rows  |
-------------------------------------------------------------------------------------------
|   1 | SORT GROUP BY                         |               |      1 |      1 | 1       |
|*  2 |  FILTER                               |               |      1 |        | 1728K   |
|   3 |   NESTED LOOPS                        |               |      1 |      1 | 1728K   |
|*  4 |    HASH JOIN                          |               |      1 |      1 | 1728K   |
|   5 |     PARTITION LIST SINGLE             |               |      1 |   6844 | 3029    |
|*  6 |      INDEX RANGE SCAN                 | PROV_IX13     |      1 |   6844 | 3029    |
|   7 |     PARTITION LIST SINGLE             |               |      1 |   5899 | 5479K   |
|*  8 |      TABLE ACCESS BY LOCAL INDEX ROWID | SERVICE       |      1 |   5899 | 5479K   |
|*  9 |       INDEX SKIP SCAN                 | SERVICE_IX8   |      1 |   4934 | 5479K   |
|  10 |    PARTITION LIST SINGLE              |               |  1728K |      1 | 1728K   |
|* 11 |     INDEX RANGE SCAN                  | CLAIM_IX7     |  1728K |      1 | 1728K   |
-------------------------------------------------------------------------------------------
```

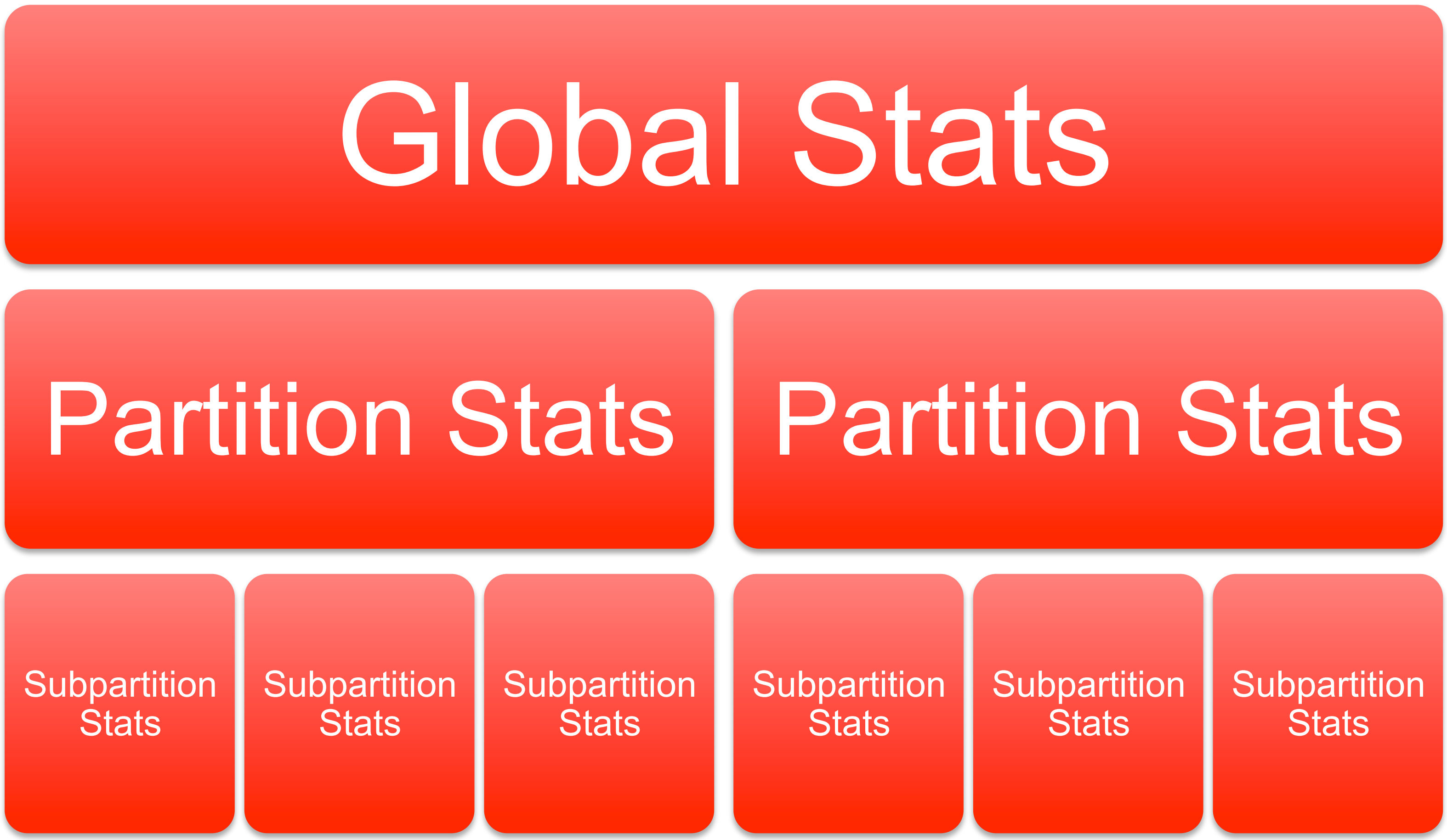
Concluding the Debugging Process

- Most execution plans are a function of bad cardinality estimates. Again watch out for cardinality Estimates of 1 !
- Poor plans with good cardinality estimates with default system statistics and init.ora settings may be candidates as a bug.
- You biggest frustration will be understanding how cardinality estimates are calculated as this is not traceable.

Managing Statistics on Partitioned Tables

ORACLE

Which Statistics to Build ?



Managing Stats on Partitioned Tables

Common Feedback

- What should my statistics gathering strategy be for partitioned tables?
- How do I maintain them and what are the challenges?
- What is best practice?
- Many conflicting opinions.

Global Statistics vs. Global and Partition Statistics

- Controlled by **`GRANULARITY`** parameter in **`DBMS_STATS.GATHER_*`**
- If partition distribution is uniform, GLOBAL statistics may be enough
- If partition distribution is skewed, GLOBAL and PARTITION are recommended
- ALL (GLOBAL/PARTITION/SUBPARTITION) is generally only gathered for non-hash subpartitions (range or list). This is the default.

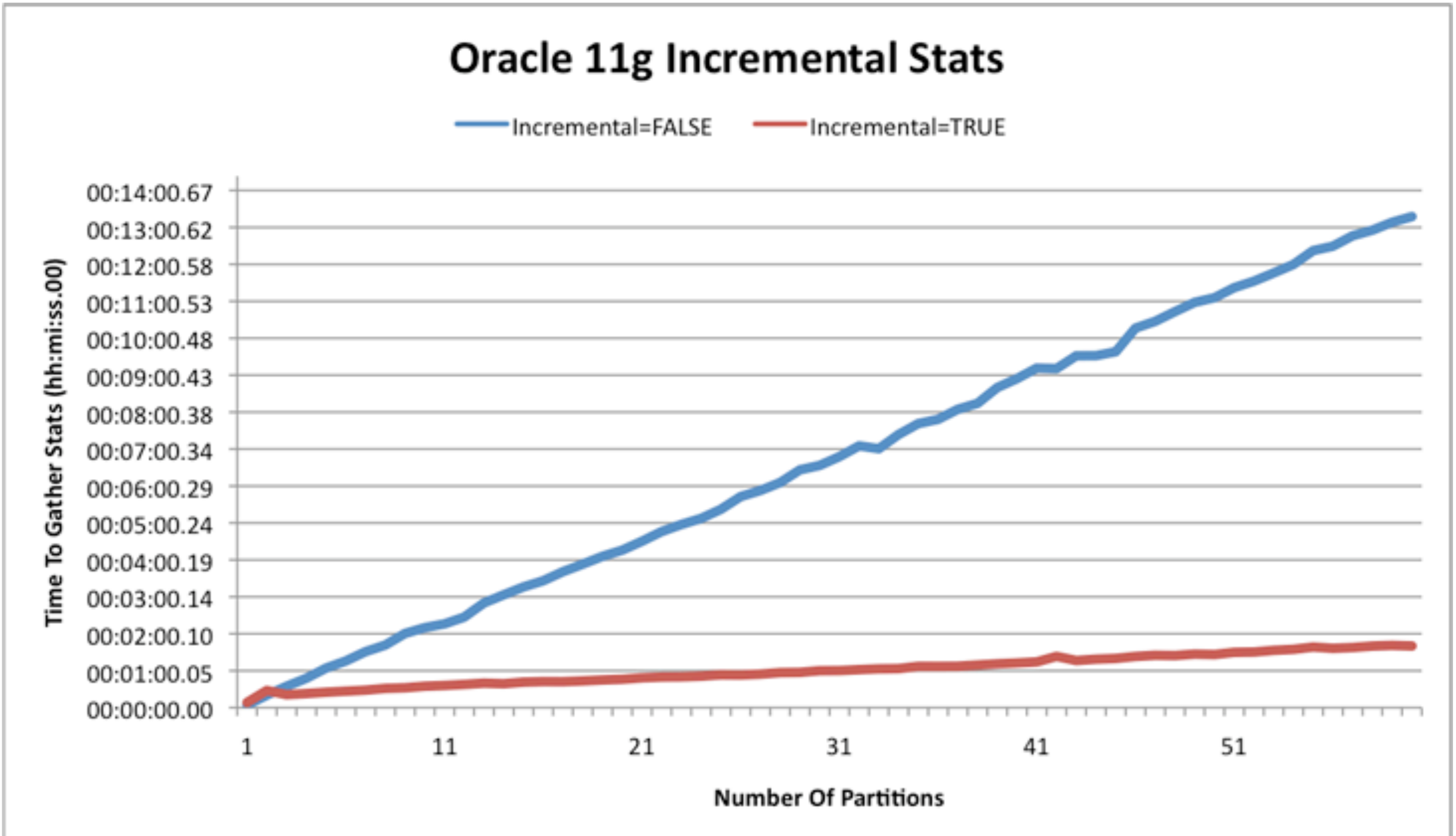
Partition Copy Statistics Introduced 10.2

- Invoked via **DBMS_STATS.COPY_TABLE_STATS**
- Allows partition statistics to be cloned from another partition
- Be aware:
  - 10.2.0.4 does not adjust the high value of the partition key column for the new partition and does not update the global high value for the partition key column
  - 11.1.0.6 does adjust the high value of the partition key column for the new partition but does not update the global high value for the partition key column
- High value partition adjustment fixed in patch for 10.2

Partition Incremental Statistics

- New in 11g
- Only available for partitioned tables
- Stores the synopsis for each partition in SYSAUX
- Global statistics are then created from the aggregation of the partition synopses. Eliminates FTS for global statistics.
- Must be manually enabled for each table
  - ```
    DBMS_STATS.SET_TABLE_PREFS(
     OWNNAME =>'SALES',
     TABNAME =>'SALES_FACT',
     PNAME   =>'INCREMENTAL',
     PVALUE  =>'TRUE');
    ```
    ```

11g Incremental Stats Time Savings



Partition Exchange Statistics

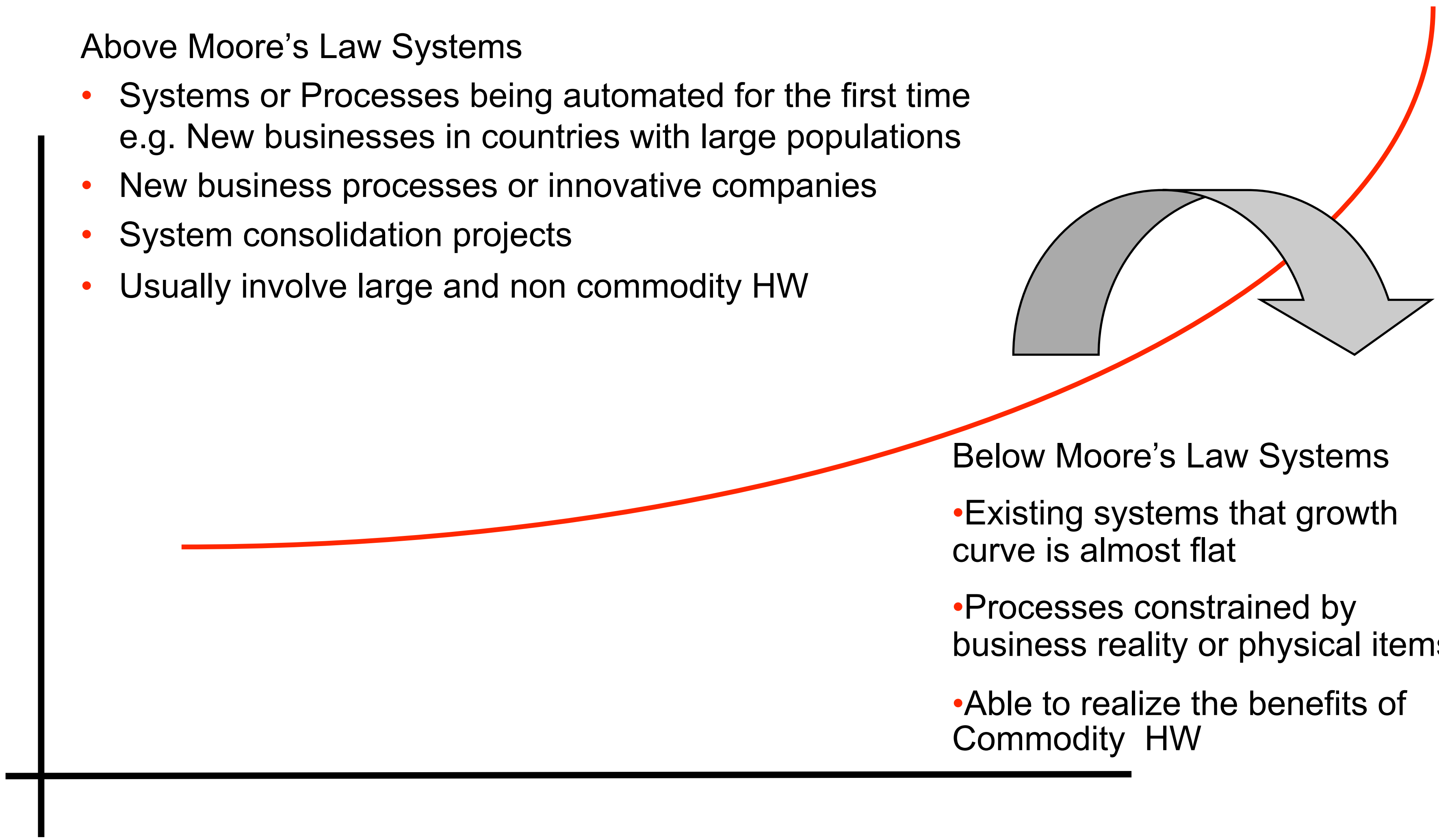
- Statistics are gathered on staging table being exchanged
- Partition exchange does not update GLOBAL statistics
  - GLOBAL statistics should be gathered later, no need to regather PARTITION stats
- GLOBAL statistics become stale when:
  - Partition exchange is used
  - Partition statistics are copied/cloned
  - Statistics are gathered with **`GRANULARITY=>'PARTITION'`**
- Incremental stats in 11g can not be used on the staging table, as it only works on partitioned tables. After the exchange, 11g incremental statistics can be used.

When to Apply the Knife to Your Data

Moore's Law Reality Check

Above Moore's Law Systems

- Systems or Processes being automated for the first time e.g. New businesses in countries with large populations
- New business processes or innovative companies
- System consolidation projects
- Usually involve large and non commodity HW



Below Moore's Law Systems

- Existing systems that growth curve is almost flat
- Processes constrained by business reality or physical items.
- Able to realize the benefits of Commodity HW

When to Apply the Knife to your Data /Workloads/Databases

- Please keep this discussion in context. If you are below the Moore's Law curve many of the techniques I will describe may not be relevant.
- Segmentation of data, workloads or databases may be driven by the desire to be below the curve and embrace the economies of commodity hardware.
- The Industry challenge is continually increasing data sizes, workloads and aggressive performance requirements.
- An obvious approach when something gets big is to start breaking it up in to smaller, more manageable pieces.
- This topic is one of the toughest design problems an Enterprise Architect has to address.

Why We Must Think Carefully Before Applying the Knife

- Remember why you used a relational database in the first place
  - Single point of truth
  - Ability to join and query data across multiple columns
  - Simplified development
  - Reduced systems and application maintenance
- As soon as you start cutting up datasets and databases the value of the database and its contents looses value.

Why We Must Think Carefully Before Applying the Knife

- Oracle has always been an advocate of single databases for the following reasons
  - The ability to answer all the queries and evolve the business within a single datastore.
  - The economies of scale in administration
  - Avoid the problems of multi database coherency

Data Segmentation

- Data segmentation is usually achieved within Oracle Databases by the use of Partitioning techniques.
- The primary uses of Partitioning are as follows
  - Large object management:
    - Makes large table(s) manageable in acceptable times e.g. Index Builds, Statistics Gathering
    - Allows fast load/purge techniques via EXCHANGE
  - Query Optimization techniques
    - Partition pruning
    - Join and Sort Optimization
  - Contention or Hot spot management
    - Right growing Indexes

Data Segmentation

- Partitioning techniques are very effective when the goals are clearly defined.
- However we must remember that by “applying the knife” there will be some implications which may impact the performance and availability of other operations within the database.

Data Segmentation

- The challenges you will face with partitioning will relate to:
  - LOCAL vs GLOBAL Indexes
    - Tables with GLOBAL indexes do not EXCHANGE quickly
    - Queries on LOCAL Indexes may require multiple probes which has an impact on query performance
  - Statistics Gathering
    - The challenges of GLOBAL vs PARTITION
  - Trading off all requirements of your partitioning strategy
    - You get to apply the knife twice (PARTITION and SUBPARTITION)
    - Do you optimize for management, query performance or scalability ?

Data Segmentation Challenges

Partition Table Goal	Design Challenges and Observations
Large Object Management	Almost mandates use of LOCAL Indexes if EXCHANGE is used for load/purge activities. Downstream query implications. Statistics Management may prove challenging.
Partition Pruning and Join/Sort Optimization	Selecting partitioning columns based on often unknown workloads. Most designs resort to time based partitions with hash to support joins.
Contention Management	Often done in a rush to satisfy INSERT contention issues in clusters. Downstream queries often neglected.

Workload Segmentation

- The consistency of response time critical applications(<5ms) is a function of the following
  - Locating the database blocks within the local instance cache
  - The ability to read/modify the database blocks without contention
- This encourages memory resident databases.
- The design challenges are:
  - What happens if the dataset is bigger than the buffer cache
  - What happens if the CPU on the host cannot sustain the workload
- In this case we need to look at workload segmentation within a cluster

Workload Segmentation

- Workload Segmentation attempts to build up memory caches of specific rows often defined by data ranges or other attributes.
- The goal of this is to ensure zero block contention and very high cache hit ratio.
- This ensures response time targets are met with the required scaling of transaction rates.

Workload Segmentation

- The designer's challenge for a segmented work load are as follows:
  - Determine on what table attributes a transaction can be routed. This is often a primary key of a driving entity e.g. cust_id, order_id, security_id etc.
  - Build data and indexing partitioning strategies to support the response time critical transactions
  - Encode routing strategies within middleware/database connection techniques
  - Enforcing the techniques with developers
- This process is always a matter of compromise and in more complex data models may be near-impossible to achieve.
- Systems that do achieve a high degree of success using these techniques tend to have very simple, denormalized schemas and are focused on executing a very small set of transactions. They do not harness the full value of the relational model on these systems.

Database Segmentation

- The last resort is to start splitting a database into multiple databases into a federation of similar databases
- This process is not for your average database and the goal here is to support a specific workload for the database.
- The types of companies that do this tend to have huge growth curves and are willing to write a great deal of middleware code.
- Examples include The World's Biggest Websites, Trading Systems, Transport ticketing systems etc.
- Database segmentation is often done for risk mitigation purposes. Companies of a critical size do not wish to stake their entire business on single database.

Database Segmentation

- The designer's challenges for a segmented database include the following
  - Segmentation of the schema
  - Routing of transactions
  - Addressing distributed transactions over multiple databases
  - Middleware logic. The middleware may require cache resident databases (TimesTen or Coherence) to cache routing information. E.g. Entity to DB translation.
- Each element of the federated database can be a clusters with its own availability/failover strategy.

Detecting and Avoiding Hiccups in your OLTP System

Detecting and Avoiding Hiccups in your OLTP System

- Over the last year, the Real World Performance Group has studied why OLTP systems may suddenly slow down and become unpredictable.
- We have many Oracle OLTP systems that are pushing over 5000 SQL statements a second.
- With this arrival rate, these systems cannot stall as this will result in back logs, queues in the application servers, often connection storms and generally a poor user experience

Detecting and Avoiding Hiccups in your OLTP System

- Over the last few years we have developed some techniques to help you debug unexplained system slowdowns.
- To demonstrate these techniques we will use the example of a recurring series of bugs we have seen over the last year.
- To perform good debugging you need good statistics. From version 10g onwards there is no excuse not to obtain good statistics.

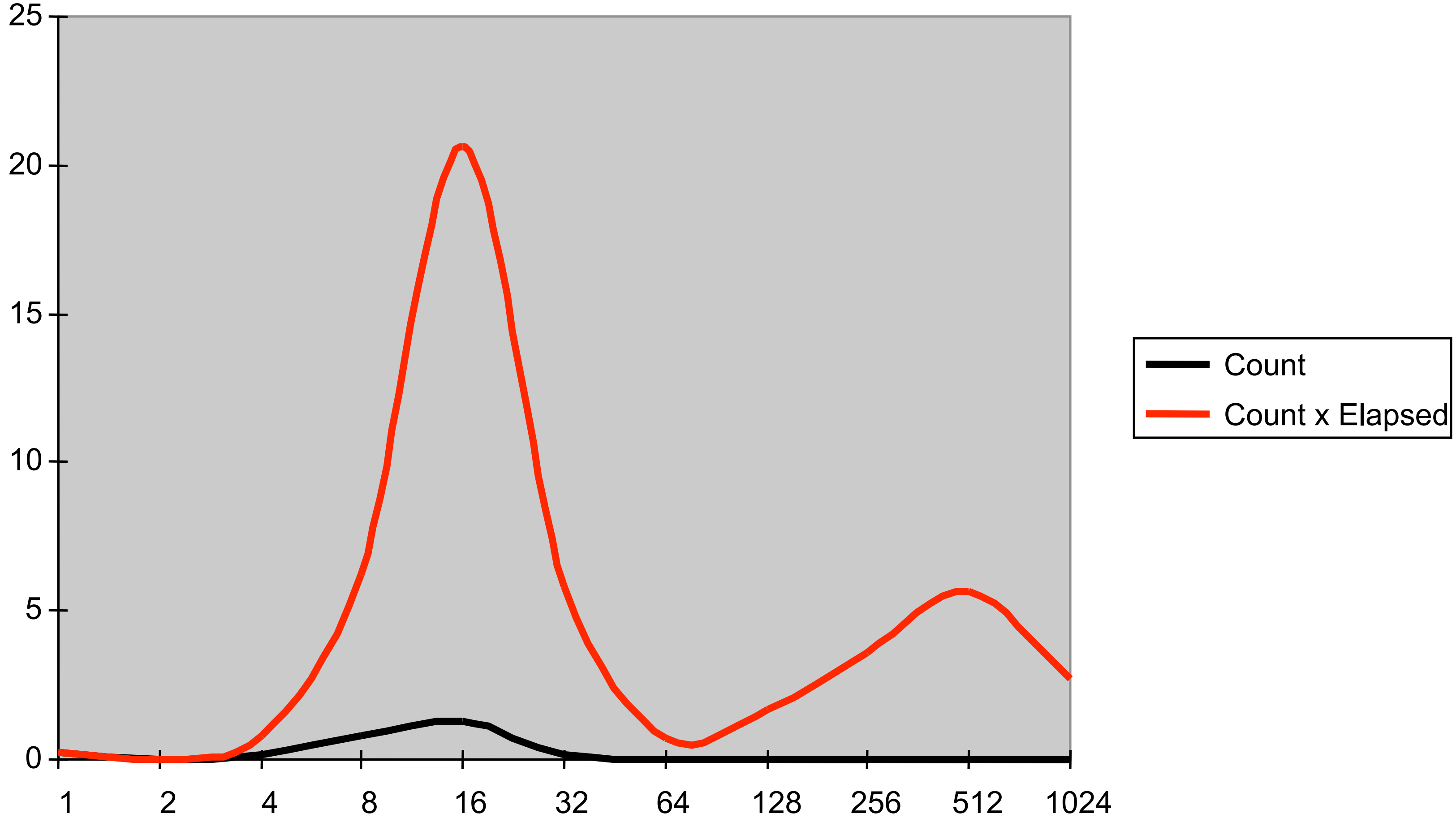
LGWR Example

- In many Oracle systems we see “log file sync” as the most dominant event
- In many cases this is the expected behavior but we started to see patterns of behavior when it should not have been the dominant event.
- In many cases the “log file sync” was seen as impacting scalability as well the user response time.
- The frustrating thing was that the average values for the “log file sync” seemed acceptable as did the actual redo log write time.
- In many cases our clients had moved redo logs to dedicated storage and even solid state devices and performance had barely improved.

Debug using v$event_histogram

```
select
event,               -- The actual wait event
wait_time_milli,  -- milli second bucket
wait_count,        -- count within bucket
wait_time_milli*wait_count -- Weighted Value
from v$event_histogram
where event in ('log file sync' )
order by 1,2
/
```

Debugging the Output



Conclusions from this Exercise

- Statistical averages can be very misleading
- Identification of the outliers provided the answers
- The bugs have since been fixed !

Hiccups Caused by Database Administration

- Statistics management
  - Statistics update
    - Cursor invalidations
    - Automatic Gathering
- Partition management
  - Exchange operations
  - Statistics management
  - Index management
- Other issues
  - Datafile extensions
  - Grants

Wrap Up

The Performance Core Disciplines Revisited

- To be good at real-world performance you must be able to do root cause analysis for the performance problem.
- Understanding the following disciplines and how they are interconnected defines the role of the performance specialist:
  - SQL execution plans
  - Buffer cache efficiency
  - Connection and cursor management
  - Contention identification and management
  - Hardware capacity planning
- Thank you see you in the next session.

Examples

Cloning Partition Stats

Step 1: Copy

```
begin
  dbms_stats.copy_table_stats(
    ownname=>user,
    tabname=>'FOO',
    srcpartname=>'P20080813',
    dstpartname=>'P20080814'
  );
end;
/
-- 10g does not automatically adjust the partition high value
-- 11g does
PARTITION_NAME  LOW_VAL              HIGH_VAL
--------------  -------------------- --------------------
P20080813       2008-08-13 00:00:00  2008-08-13 23:59:59
P20080814       2008-08-13 00:00:00  2008-08-13 23:59:59
```

Cloning Partition Stats

Step 2: Adjust partition key column high value (10g)

```
declare
  srec dbms_stats.statrec;
  datevals dbms_stats.datearray;
begin
  srec.eavs := 0;
  srec.chvals := null;
  datevals:= dbms_stats.datearray(
    to_date('2008-08-14 00:00:00','yyyy-mm-dd hh24:mi:ss'),
    to_date('2008-08-14 23:59:59','yyyy-mm-dd hh24:mi:ss'));
  srec.bkvals := dbms_stats.numarray(0,1);
  srec.epc := 2;
  dbms_stats.prepare_column_values(
    srec=>srec,
    datevals=>datevals
  );
  dbms_stats.set_column_stats(
    ownname=>user,
    tabname=>'FOO',
    colname=>'PART_KEY',
    partname=>'P20080814',
    srec=>srec
  );
end;
/
```

Cloning Partition Stats

Step 3: Adjust Global (Table) High Value

```
declare
  srec dbms_stats.statrec;
  datevals dbms_stats.datearray;
begin
  srec.eavs := 0;
  srec.chvals := null;
  datevals:= dbms_stats.datearray(
    to_date('2008-08-12 00:00:00','yyyy-mm-dd hh24:mi:ss'),
    to_date('2008-08-14 23:59:59','yyyy-mm-dd hh24:mi:ss'));
  srec.bkvals := dbms_stats.numarray(0,1);
  srec.epc := 2;
  dbms_stats.prepare_column_values(
    srec=>srec,
    datevals=>datevals
  );
  dbms_stats.set_column_stats(
    ownname=>user,
    tabname=>'FOO',
    colname=>'PART_KEY',
    srec=>srec,
    distcnt=> 3*86400 --hardcoded value. Use get_column_stats for actual value
end;
/
```

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.