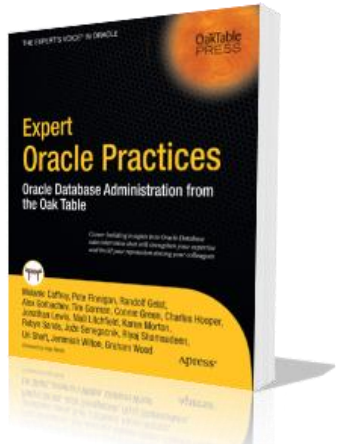
Oracle Optimizer System Statistics Update 2018

Randolf Geist

oracle-performance.de

oracle-performance.de

- Independent consultant
  - Performance Troubleshooting
  - In-house workshops
    - Cost-Based Optimizer
    - Performance By Design
- Oracle ACE Director Alumni
- Member of OakTable Network



ORACLE ACE Director

2009-2016 Alumni

OakTable.net

Agenda (1)

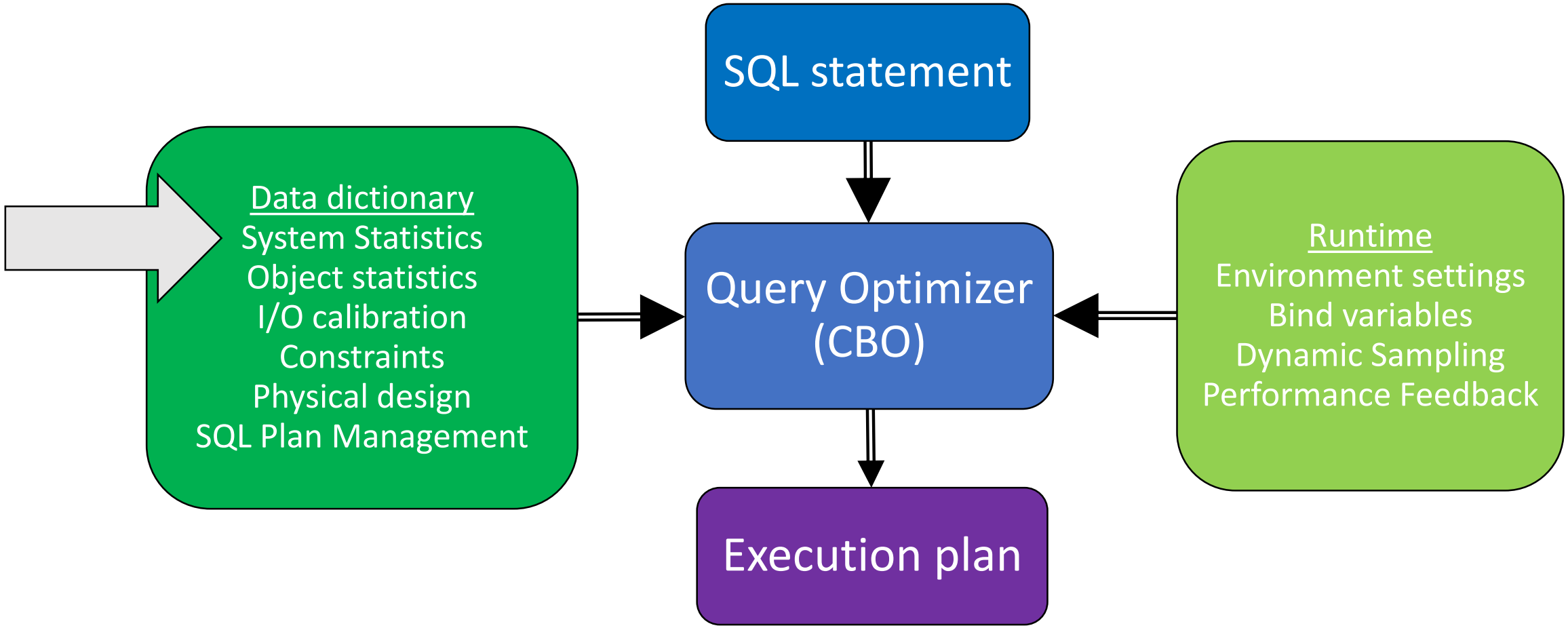
- What are System Statistics?
- History of System Statistics
- How System Statistics work
- Practical usage of System Statistics
- New features of System Statistics

Agenda (2)

- What is the I/O calibration?
- How the I/O calibration works
- New features of the I/O calibration

What are System Statistics?

What are System Statistics?



What are System Statistics?

System Statistics can provide information about the CPU and I/O capabilities of the system as well as I/O patterns of the database:

- The disk seek time (IOSEEKTIM)
- The disk transfer speed (IOTFRSPEED)
- The CPU speed (CPUSPEED[NW])
- The time a single/multi block read takes (SREADTIM / MREADTIM)
- The average number of blocks read per multi-block read request (MBRC)
- Maximum I/O throughput of the system (MAXTHR)
- I/O throughput per Parallel Execution Server (SLAVETHR)

What are System Statistics?

System Statistics can provide information about the CPU and I/O capabilities of the system as well as I/O patterns of the database:

- The disk seek time (IOSEEKTIM)
- The disk transfer speed (IOTFRSPEED)
- The CPU speed (CPUSPEEDNW)

NOWORKLOAD System Statistics

- The time a single/multi block read takes (SREADTIM / MREADTIM)
- The average number of blocks read per multi-block read request (MBRC)
- Maximum I/O throughput of the system (MAXTHR)
- I/O throughput per Parallel Execution Server (SLAVETHR)

What are System Statistics?

System Statistics can provide information about the CPU and I/O capabilities of the system as well as I/O patterns of the database:

- The disk seek time (IOSEEKTIM)
- The disk transfer speed (IOTFRSPEED)
- The CPU speed (CPUSPEED)
- The time a single/multi block read takes (SREADTIM / MREADTIM)
- The average number of blocks read per multi-block read request (MBRC)
- Maximum I/O throughput of the system (MAXTHR)
- I/O throughput per Parallel Execution Server (SLAVETHR)

WORKLOAD System Statistics

History of System Statistics

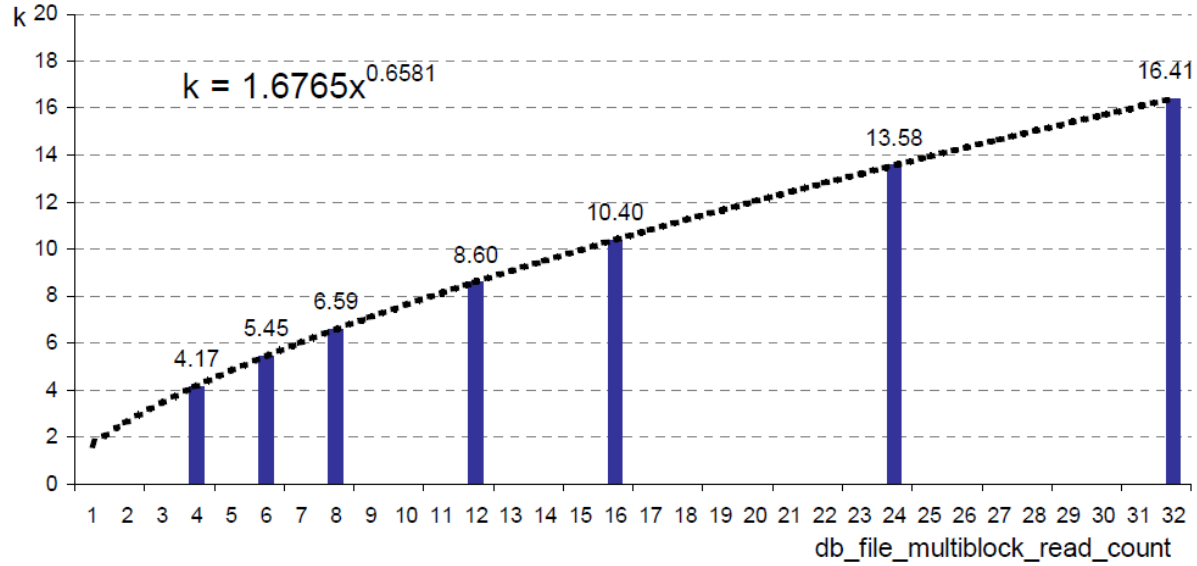
History of System Statistics – Oracle 7

- Cost based optimizer (CBO) introduced in Oracle 7
- Traditional I/O costing – selecting the plan with the least I/O cost
- Costing based on **counts** of single and multi block read requests
- But single and multi block read requests were treated the same

⇒This tended to favour Full Table Scans

History of System Statistics

- Cost of Full Table Scan was based on DB_FILE_MULTIBLOCK_READ_COUNT (MBRC) setting
- Higher setting => Lower costs
- Costing model included an "adjusted" (lowered) MBRC used internally



Tablescan cost factor k and db_file_multiblock_read_count

Source: „A look under the hood of the CBO" (2004) by Wolfgang Breitling

History of System Statistics – Oracle 8i

To address the “favour Full Table Scans” issue Oracle 8i introduced two (infamous) parameters:

OPTIMIZER_INDEX_COST_ADJ

Allows adjusting the cost of index access paths. Usually used to lower the cost, favouring index access

OPTIMIZER_INDEX_CACHING

Allows lowering costs of certain access paths: Inner row source index access of Nested Loop join and IN list operators. The idea behind is modifying the cost calculation by considering caching effects – default assumption of the CBO is every block access leads to physical I/O

History of System Statistics – Oracle 8i

Still there were problems:

- What is the “correct” non-default setting of the OPTIMIZER_INDEX* parameters?
- DB_FILE_MULTIBLOCK_READ_COUNT setting isn’t necessarily a good reflection of the actual MBRC used at runtime (caching of blocks, extent boundaries etc.)
- Too low settings of OPTIMIZER_INDEX_COST_ADJ could result in access paths with same costs, CBO choses by **index name (!)** then

History of System Statistics – Oracle 9i

Enter System Statistics, introduced with Oracle 9i, main objectives:

- Configure single and multi block read relative costs by introducing a time component for each
- Adjust costing to individual environment by measurement
- Adjust cost of Full Table Scans instead of Index access paths
- Introduce a CPU costing component including predicate reordering

History of System Statistics – Oracle 10g

Oracle 10g introduced default System Statistics, main objectives:

- Increase the cost of Full Table Scans compared to traditional I/O costing
- Further soften the effect of very high DB_FILE_MULTIBLOCK_READ_COUNT settings
- Decouple runtime MBRC from costing MBRC

⇒Affected potentially every execution plan

⇒Caused a lot of headaches in the beginning

How System Statistics work

How System Statistics work

Configure single and multi block read relative costs by introducing a time component for each (SREADTIM / MREADTIM)

=> Addresses the problem of treating single and multi block access the same

Assumption (and this is a built-in validity / sanity check) is that multi block reads have to take longer than single block reads – not always true in reality (SAN read ahead caching effects)

How System Statistics work

Adjust costing to individual environment by measurement

=> Addresses the question:
"What is the correct setting for my environment"?

How System Statistics work

Adjust cost of Full Table Scans instead of Index access paths

=> Addresses the “Same (small) cost choose index by name issue”

How System Statistics work

Introduce a CPU costing component (CPUSPEED[NW]) including predicate reordering

=> Improves the performance by evaluating less CPU intensive expressions first and considering CPU in general as part of the cost calculation

How System Statistics work

Takes (measured) MBRC value into account instead of using the DB_FILE_MULTIBLOCK_READ_COUNT parameter as part of the cost calculation

=> Addresses the problem of difference between parameter setting and real MBRC at runtime

How System Statistics work

Total system and Parallel Execution Server throughput (MAXTHR and SLAVETHR) can be used to configure Parallel Execution cost calculation more realistically

=> Addresses shortcomings / default assumptions of the CBO cost model, for example default assumption of Parallel Execution infinite scaling

Practical usage of System Statistics

Practical usage of System Statistics

- Improved CBO cost calculation treating single and multi block reads differently
- CPU cost component, advanced features like predicate reordering
- Aim of default System Statistics introduced in 10g: Make the CBO cost calculation **more index friendly** by default

Default System Statistics – increased FTS cost

Full Table Scan costs without System Statistics MBRC = 8:

```
-----------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost  |
-----------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1518 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |       |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1518 |
-----------------------------------------------------------
```

Full Table Scan costs with default System Statistics MBRC = 8:

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  2709   (0)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  2709   (0)| 00:00:33 |
-------------------------------------------------------------------------------
```

Default System Statistics – high MBRC

Full Table Scan costs without System Statistics MBRC = 128:

```
-----------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost  |
-----------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |   245 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |       |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |   245 |
-----------------------------------------------------
```

Full Table Scan costs with default System Statistics MBRC = 128:

```
-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1732   (0)| 00:00:21 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1732   (0)| 00:00:21 |
-------------------------------------------------------------------------
```

Practical usage of System Statistics

- It’s the ratio between SREADTIM and MREADTIM together with MBRC that are most relevant for the cost calculation – not the absolute values of SREADTIM / MREADTIM

⇒Changing the ratio between SREADTIM and MREADTIM makes the cost calculation either more index or full table scan friendly

⇒Decreasing / increasing MBRC does the same

Practical usage of System Statistics

- In principle you could achieve the same using OPTIMIZER_INDEX_COST_ADJ, but:
  - OPTIMIZER_INDEX_COST_ADJ adjusts index costs. System Statistics adjust Full Table Scan costs instead
  - There was no official way of **measuring** reasonable values for OPTIMIZER_INDEX_COST_ADJ, whereas System Statistics offer an official interface for measuring and therefore represent the CPU and I/O capabilities of the individual environment

Practical usage of System Statistics

More advanced features of (WORKLOAD) System Statistics allow configuring Parallel Execution costing. Without System Statistics:

- The CBO assumes infinite scaling of Parallel Execution (quite unrealistic) – the higher the degree, the lower the cost
- Assumes per default that Parallel Execution scales at a factor of 0.9 (90%) of serial execution

Practical usage of System Statistics

Parallel Execution scaling factor can be lowered by SLAVETHR parameter. You can't make it scale better than 0.9 – in that case SLAVETHR will be ignored and the default value / calculation applies

- Allows configuring the system less Parallel Execution friendly but more serial index access friendly
- Default value of SLAVETHR is:

  0.9 * MBRC * db_block_size / MREADTIM * 1000 bytes per second

Default costing of Parallel Execution

```
select max(n1) from t1;
```

```
-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   278   (0)| 00:00:02 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   278   (0)| 00:00:02 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   278   (0)| 00:00:02 |
-------------------------------------------------------------------------------
```

Increased costing of Parallel Execution

```
Increased parallel cost at same degree when SLAVETHR is lower than default value
-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   800   (0)| 00:00:05 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   800   (0)| 00:00:05 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   800   (0)| 00:00:05 |
-------------------------------------------------------------------------------
```

Practical usage of System Statistics

MAXTHR can be used to limit default infinite scaling of Parallel Execution

```
select max(n1) from t1;


Serial cost:
-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     5 |  1280   (3)| 00:00:07 |
|   1 |  SORT AGGREGATE    |      |     1 |     5 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 40000 |   195K|  1280   (3)| 00:00:07 |
-------------------------------------------------------------------------
```

Default unlimited scaling of Parallel Execution

```
Parallel(T1 42) cost without MAXTHR set
-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |    33   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|    33   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|    33   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Limited scaling of Parallel Execution

```
Parallel(T1 42) cost with MAXTHR set
-------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     1 |     5 |   250   (0)| 00:00:02 |
|   1 |  SORT AGGREGATE          |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR         |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM)   | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE       |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR   |          | 40000 |   195K|   250   (0)| 00:00:02 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   250   (0)| 00:00:02 |
-------------------------------------------------------------------------------
```

New features of System Statistics

New features of System Statistics

- Not many new features / changes were introduced in recent years

- Most changes were actually about fixing bugs:
  - 11.2.0.1 / 11.2.0.2 were affected by a nasty bug that measured completely unrealistic values for SREADTIM / MREADTIM when measuring WORKLOAD System Statistics
  - For SLAVETHR and MAXTHR an odd factor was used as part of the cost calculation (likewise the unit used)
  - All these were fixed as of 11.2.0.3 / 12.1.0.2

New features of System Statistics

- 12c (also available in 11.2.0.4) introduces “EXADATA” System Statistics

- These are actually a flavour of (default) NOWORKLOAD System Statistics

- Main reason for introduction:
  - CBO cost calculation by default doesn’t take into account the potential speed up of Exadata Smart Scans

New features of System Statistics

The concept of “EXADATA” System Statistics is simple:

- It just sets the MBRC value to the value of the (undocumented) “_db_file_exec_read_count” parameter
- This parameter is only different from “_db_file_optimizer_read_count” (used for cost calculation) when the official parameter “db_file_multiblock_read_count” is left unset

⇒“EXADATA” mode makes only a difference in cost calculation when “db_file_multiblock_read_count” **is left unset**

⇒Not limited to “EXADATA”, can be used in any environment

“EXADATA” System Statistics

Sample Full Table Cost with default System Statistics and "db_file_multiblock_read_count" left unset

```
 ---------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  2716   (1)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  2716   (1)| 00:00:33 |
---------------------------------------------------------------------------
```

Sample Full Table Cost with EXADATA System Statistics, which sets MBRC to 128 in most cases when using 8 KB default block size and leaving "db_file_multiblock_read_count" unset

```
---------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1738   (1)| 00:00:21 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1738   (1)| 00:00:21 |
---------------------------------------------------------------------------
```

New features of System Statistics

- „EXADATA“ System Statistics should only be used if Full Table Scans are really supposed to be favoured – not necessarily the case with mixed workload OLTP / batch applications

- Due to the way the NOWORKLOAD System Statistics calculate SREADTIM and MREADTIM the cost difference is not that significant when using “EXADATA” System Statistics

⇒If you really want to favour Full Table Scans manually setting WORKLOAD System Statistics value of SREADTIM and MREADTIM might be more useful

What is the I/O calibration?

What is the I/O calibration?

The I/O calibration describes I/O capabilities of the database

$\Rightarrow$Sounds familiar?

What is the I/O calibration?

- Gets used as part of the “Auto DOP” feature introduced in Oracle 11.2
- “Auto DOP” attempts to determine automatically if a statement should be executed using Parallel Execution and if yes at what degree
- Calculation mainly based on **I/O throughput** determined by I/O calibration

⇒But I/O throughput figures are already available via System Statistics!

Why then I/O calibration?

- Measuring System Statistics potentially affects all execution plans system wide

- I/O calibration is explicitly based on asynchronous I/O and only works if that is available and enabled

⇒You can use and configure “Auto DOP” without the risk of measuring System Statistics by making use of “I/O Calibration”

How I/O calibration works

How I/O calibration works

I/O Calibration measures the following:

- Maximum number of I/O read requests per second (MAX_IOPS)
- Maximum Megabytes per second of I/O throughput (MAX_MBPS)
- Maximum Megabytes per second of I/O throughput a single process can sustain (MAX_PMBPS)
- Latency in milliseconds (LATENCY)

How System Statistics work

System Statistics can provide information about the CPU and I/O capabilities of the system as well as I/O patterns of the database:

- The time a single/multi block read takes (SREADTIM / MREADTIM)
- The disk seek time (IOSEEKTIM)
- The disk transfer speed (IOTFRSPEED)
- The CPU speed (CPUSPEED[NW])
- The average number of blocks read per multi-block read request (MBRC)
- **Maximum I/O throughput of the system (MAXTHR)**
- **I/O throughput per Parallel Execution Server (SLAVETHR)**

How I/O calibration works

I/O Calibration measures the following:

- Maximum number of I/O read requests per second (MAX_IOPS)
- **Maximum Megabytes per second of I/O throughput (MAX_MBPS)**
- **Maximum Megabytes per second of I/O throughput a single process can sustain (MAX_PMBPS)**
- Latency in milliseconds (LATENCY)

How I/O calibration works

- I/O Calibration is part of the Resource Manager
- No API available comparable to System Statistics
- Only a single call to DBMS_RESOURCE_MANAGER.CALIBRATE_IO available
  - ⇒You need to specify the “approximate number of physical discs” and the “maximum latency” in the call
  - ⇒At least “maximum latency” seems to be used to control the artificial load generated

How I/O calibration works

- DBMS_RESOURCE_MANAGER.CALIBRATE_IO spawns CPU_COUNT slaves (on each RAC instance in case of RAC) and generates **artificial** workload reading from **all** datafiles => Environment should be idle

⇒If you happen to have a mixture of storage this will average across them

- The artificial workload is a sequence of:
  - Random single block reads (multiple sessions)
  - Sequential multi block reads (multiple sessions)
  - Sequential multi block reads (single session)

How I/O calibration works

- The calibration process can be monitored in V$IO_CALIBRATION_STATUS
- The results are available afterwards from DBA_RSRC_IO_CALIBRATE (based on underlying table SYS.RESOURCE_IO_CALIBRATE$)
- Only relevant value for Auto DOP calculation as of today is **MAX_PMBPS** (Maximum Megabytes per second of I/O throughput a single process can sustain)

How I/O calibration works

Known problems with I/O Calibration:

- Instance restart required to take effect
- Measured values might not represent the storage capabilities
  - In particular the measured latency might be too low, related to how asynchronous I/O and corresponding wait events in Oracle work
  - But also the measured throughput might be too low and the measured IOPS too high
  - Recommendation to set values manually (in versions below 12c) on systems with fast storage like Exadata (MOS document 1269321.1) – which actually means manipulating SYS.RESOURCE_IO_CALIBRATE$ via DML, no API available

How I/O calibration works

- **MAX_PMBPS** (Maximum Megabytes per second of I/O throughput a single process can sustain) controls a new calculation that turns the cost into a new TIME estimate
- The relationship between COST and TIME changes with I/O calibration results
- So far it used to be: TIME equals COST multiplied by SREADTIM
- Or in other words: COST is TIME expressed in number of single block reads

How I/O calibration works

```
TIME represents cost multiplied by SREADTIM without I/O calibration, in
this case here 2716 * 12 ms equals approx. 32.6 seconds

-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  2716   (1)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T    | 10000 |  2716   (1)| 00:00:33 |
-----------------------------------------------------------------------

New calculation of TIME in the presence of I/O calibration results

-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  2716   (1)| 00:00:06 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T    | 10000 |  2716   (1)| 00:00:06 |
-----------------------------------------------------------------------
```

How I/O calibration works

This **new TIME** estimate is used to drive the decisions when the new "Auto DOP" feature gets used:

⇒Should the statement get executed using Parallel Execution?

Configurable via PARALLEL_MIN_TIME_THRESHOLD
(default 10 seconds)

⇒If it should use Parallel Execution, what degree should be used?

Mainly controlled via MAX_PMBPS value (pre 12c)

I/O calibration new features

I/O calibration new features

12c introduced default I/O calibration results: Assumes the 200 MB per second MAX_PMBPS value recommended in MOS note 1269321.1

⇒You no longer need to run I/O calibration to make use of “Auto DOP” feature

⇒In 11.2 “Auto DOP” calculation was skipped when I/O calibration was missing, fallback to default (manual) parallel degree

⇒Significant change in behavior, **can’t** be reverted using OPTIMIZER_FEATURES_ENABLE, in particular (wrong) usage of PARALELL statement level hint will cause problems after upgrade

I/O calibration new features

select /*+ parallel */ * from t2 in 11.2.0.4 without I/O calibration results, fallback to default parallel degree, which is 8 in this case here:

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  1000K|   113M|   651   (1)| 00:00:08 |
|   1 |  PX COORDINATOR      |          |       |       |            |          |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  1000K|   113M|   651   (1)| 00:00:08 |
|   3 |    PX BLOCK ITERATOR |          |  1000K|   113M|   651   (1)| 00:00:08 |
|   4 |     TABLE ACCESS FULL| T2       |  1000K|   113M|   651   (1)| 00:00:08 |
-------------------------------------------------------------------------------

Note
-----
   - automatic DOP: skipped because of IO calibrate statistics are missing
```

I/O calibration new features

select /*+ parallel */ * from t2 in 12.2.0.1 without using I/O calibration results, using default values now, computes a degree of 2 here (**same behavior even with OFE set to 11.2.0.4!**):

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  1000K|   113M|  2600   (1)| 00:00:01 |
|   1 |  PX COORDINATOR      |          |       |       |            |          |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  1000K|   113M|  2600   (1)| 00:00:01 |
|   3 |    PX BLOCK ITERATOR |          |  1000K|   113M|  2600   (1)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| T2       |  1000K|   113M|  2600   (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

Automatic Degree of Parallelism Information:

--

- Degree of Parallelism of 2 is derived from scan of object CBO_TEST.T2

I/O calibration new features

12c introduced new parameter PARALLEL_DEGREE_LEVEL that allows configuring the Parallel degree calculated from MAX_PMBPS

⇒You no longer need to manually modify MAX_PMBPS in SYS.RESOURCE_IO_CALIBRATE$ if you are not satisfied with the DOP calculation results

⇒Default value is 100, lower and higher values change the degree calculation accordingly

Oracle Optimizer System Statistics Update 2018

Thank you!

Q&A

How to configure System Statistics

How to configure System Statistics

- There is an extensive interface available for configuring System Statistics via the DBMS_STATS package

  Allows measuring, exporting, importing, reading, deleting, restoring and manually setting of System Statistics

How to configure System Statistics

- There are two flavours of System Statistics:
  - NOWORKLOAD
  - WORKLOAD

How to configure System Statistics

NOWORKLOAD System Statistics only consist of a limited set of parameters:

- The disk seek time (IOSEEKTIM)
- The disk transfer speed (IOTFRSPEED)
- The CPU speed (CPUSPEEDNW)
- The average number of blocks read per multi-block read request (MBRC) (Only in “EXADATA” mode, will be covered later)

The default System Statistics introduced in Oracle 10g are NOWORKLOAD System Statistics

How to configure System Statistics

It is called NOWORKLOAD System Statistics because when measuring via DBMS_STATS.GATHER_SYSTEM_STATS the database will generate an artificial I/O load – so the measurement isn't based on the actual workload of the database but on the artificial I/O patterns generated actively by the call to DBMS_STATS.GATHER_SYSTEM_STATS

Therefore this measurement should be performed while the database is "idle"

How to configure System Statistics

WORKLOAD System Statistics consist of a broader set of parameters:

- The time a single/multi block read takes (SREADTIM / MREADTIM)
- The CPU speed (CPUSPEED)
- The average number of blocks read per multi-block read request (MBRC)
- Maximum I/O throughput of the system (MAXTHR)
- I/O throughput per Parallel Execution Server (SLAVETHR)

How to configure System Statistics

It is called WORKLOAD System Statistics because when measuring via DBMS_STATS.GATHER_SYSTEM_STATS the database the measurement is based on the actual workload of the database and **no** artificial I/O patterns are generated actively by the call to DBMS_STATS.GATHER_SYSTEM_STATS

Therefore this measurement needs to be performed while the database is **active**

Recommendations regarding System Statistics

Recommendations regarding System Statistics

Problem: Very different statements regarding System Statistics over time

In the beginning when introduced in Oracle 9i, Oracle highly recommended measuring WORKLOAD System Statistics

In the meanwhile Oracle says **default System Statistics should be sufficient**, except you already have good experience with the use of non-default System Statistics or are willing to spend time on measuring and in particular testing them during some suitable activity, like system or hardware upgrade

Christian Antognini considers them as initialisation parameters and therefore still recommends measuring them in “Troubleshooting Oracle Performance”, but goes into more details how to do that

Recommendations regarding System Statistics

Problem: Measuring System Statistics isn't really producing constant results

Depending on the activity and gathering period the values measured might be pretty different

⇒Don't establish a regular "Gather System Statistics" job, risk of plan instability

One possible approach: Take several measures at different point in times but don't actively use instead store them in a backup table and evaluate them manually to determine and judge which values could be reasonable for your environment, then set them manually