

# Understanding the different modes of System Statistics and effects of multiple block sizes

Randolf Geist

# Who am I

---

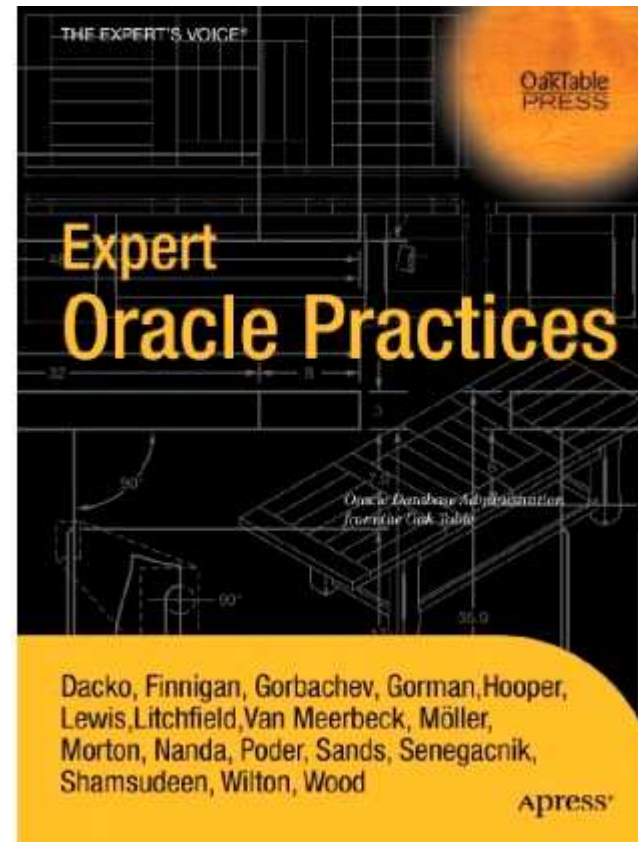
- Freelance Consultant
- Located in Germany
- Oracle ACE, OCP 8i, 9i, 10g
- Member of the OakTable Network
- My Blog: Oracle related stuff



# Who am I

---

Co-author of  
the forthcoming  
OakTable book  
"Expert  
Oracle  
Practices"



# Highlights

---

- Introduction to the Cost-Based Optimizer (CBO)
- Evolution of the CBO – traditional costing / System Statistics
- Different modes of System Statistics and implications
- Effects of multiple block sizes on the cost calculation
- Questions and Answers

# Understanding System Statistics

---

## Introduction to the Cost-Based Optimizer (CBO)

# Introduction to the Cost-Based Optimizer (CBO)

---

- Introduced with Oracle 7
- Starting with Oracle 10g the Rule-Based Optimizer (RBO) is no longer supported
- The purpose of the optimizer is to determine „how“ to execute a SQL statement (find an execution plan for a SQL statement)
- The CBO calculates the „cost“ of different possible execution plans for a SQL statement
- The „cost“ is based on applying a model to available statistics

# Introduction to the Cost-Based Optimizer (CBO)

---

- What is meant by „cost“?
  - Cost is Time: Expressed in number of Single Block Reads
- If you know the time a Single Block Read takes you can turn the cost into an actual time
- See column „TIME“ introduced with System Statistics in Execution Plans

# Understanding System Statistics

---

## Evolution of the CBO

# Evolution of the CBO

## Traditional Costing

---

- Counts the number of I/O requests
- I/O requests can be either single block reads (typically index range scans or table access by ROWID) or multi block reads (typically Full Table Scans (FTS) or Index Fast Full Scans (IFFS))
- CPU consumption is not part of the calculation, purely I/O based

# Evolution of the CBO

## Traditional Costing

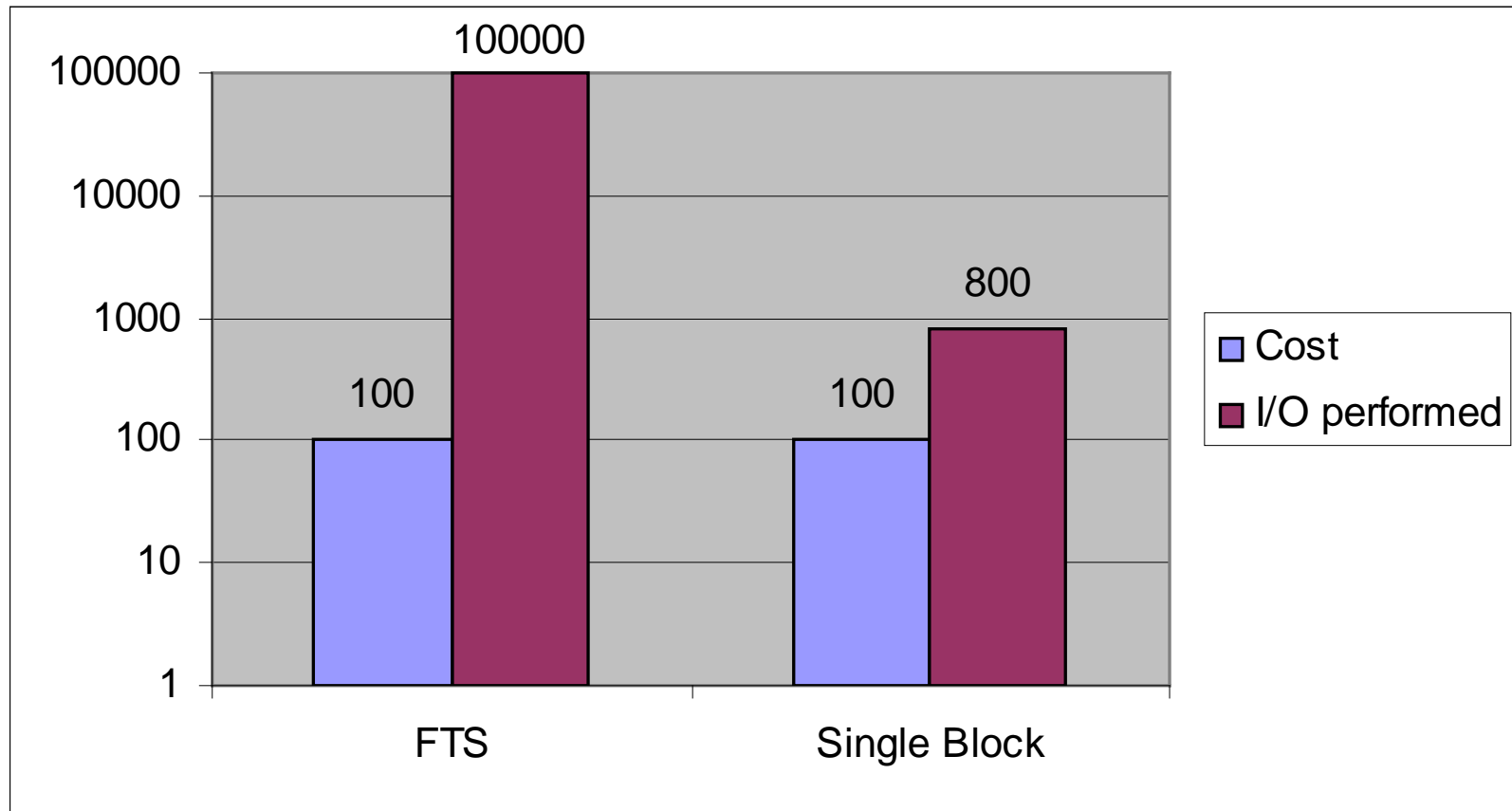
---

- Potential issue: A single block read is treated the same as a multi block read
- Simplified example: A 100 MB table segment is scanned using an FTS. With each multi block read request  $128 * 8\text{KB}$  blocks = 1MB are read. This counts as **100** I/O requests
- Reading  $100 * \text{single } 8\text{KB}$  blocks = 800KB from the same table segment also counts as **100** I/O requests

# Evolution of the CBO

## Traditional Costing

---



# Evolution of the CBO

## Traditional Costing

---

- So in this example processing 800KB and 100MB will lead to the same „cost“ of 100
- This approach tends to be biased towards Full Table Scan operations, in particular when the number of blocks read per multi block read request is large
- But from an I/O perspective you usually want the system to perform the FTS as fast as possible => use large multi block read requests => cost calculation will favor FTS operation

# Evolution of the CBO Traditional Costing

---

- Oracle introduced two (in)famous parameters in Oracle 8 to address these issues
- OPTIMIZER\_INDEX\_COST\_ADJ (OICA) default 100
- OPTIMIZER\_INDEX\_CACHING (OIC) default 0
- These can be set to non-default values to scale the cost of index access paths

# Evolution of the CBO

## Traditional Costing

---

- Setting OICA to values less than 100 reduces index access paths costs accordingly
- Setting OICA to values greater than 100 increases index access paths costs
- Setting OIC to values greater than 0 reduces the cost of particular operations (e.g. inner table index access cost of Nested Loop)

# Evolution of the CBO

## Traditional Costing

---

- Potential issues with these parameters:
  - What is the „correct“ setting?
  - If used to reduce the cost of index access paths in order to favor index access instead of Full Table Scan, the scaled down costs might lead to situations where multiple access paths get the same cost and the optimizer potentially chooses the „wrong“ index (simply by alphabetical order of index names)

# Evolution of the CBO System Statistics

---

- Enter System Statistics
- Were introduced in Oracle 9i
- System Statistics address two main issues:
  - Multi block and single block read requests will be treated differently in terms of costing
  - The cost calculation will be extended to include a CPU component attempting to cover CPU intensive operations

# Evolution of the CBO System Statistics

---

- System Statistics provide information about the capabilities of the system the database is running on, e.g.:
  - The time a single/multi block read takes
  - The disk seek time
  - The disk transfer speed
  - The CPU speed
  - The average number of blocks read per multi-block read request etc.



# Evolution of the CBO System Statistics

---

- Simplified example: A 100MB table segment is scanned using an FTS. With each multi block read request  $128 * 8\text{KB}$  blocks = 1 MB are read.
- Now with System Statistics the CBO „knows“ that a multi block read request takes a particular time, e.g. assumes reading  $128 * 8 \text{ KB}$  blocks = 1MB takes 100 ms

# Evolution of the CBO System Statistics

---

- So the FTS will take  $100 * 100 \text{ ms} = 10,000 \text{ ms} = 10 \text{ s}$  according to the calculation
  - Turning this time into the traditional cost is simply done by dividing this time by the time it takes to perform a single block read request, which is also „known“ to the CBO
  - Assumption (Example): Single block read request takes  $10 \text{ ms} \Rightarrow$  above FTS cost =  $10,000 \text{ ms} / 10 \text{ ms} = 1,000$

# Evolution of the CBO System Statistics

---

- Reading the  $100 * 8\text{KB}$  blocks via single block read request will take according to the assumed read time  $100 * 10 \text{ ms} = 1,000 \text{ ms} = 1\text{s}$ .
- Turning this time into the cost can again be done by dividing this again by  $10 \text{ ms}$ , which results in a cost of 100.
- This means that single read block costs basically are left unchanged with  
System Statistics

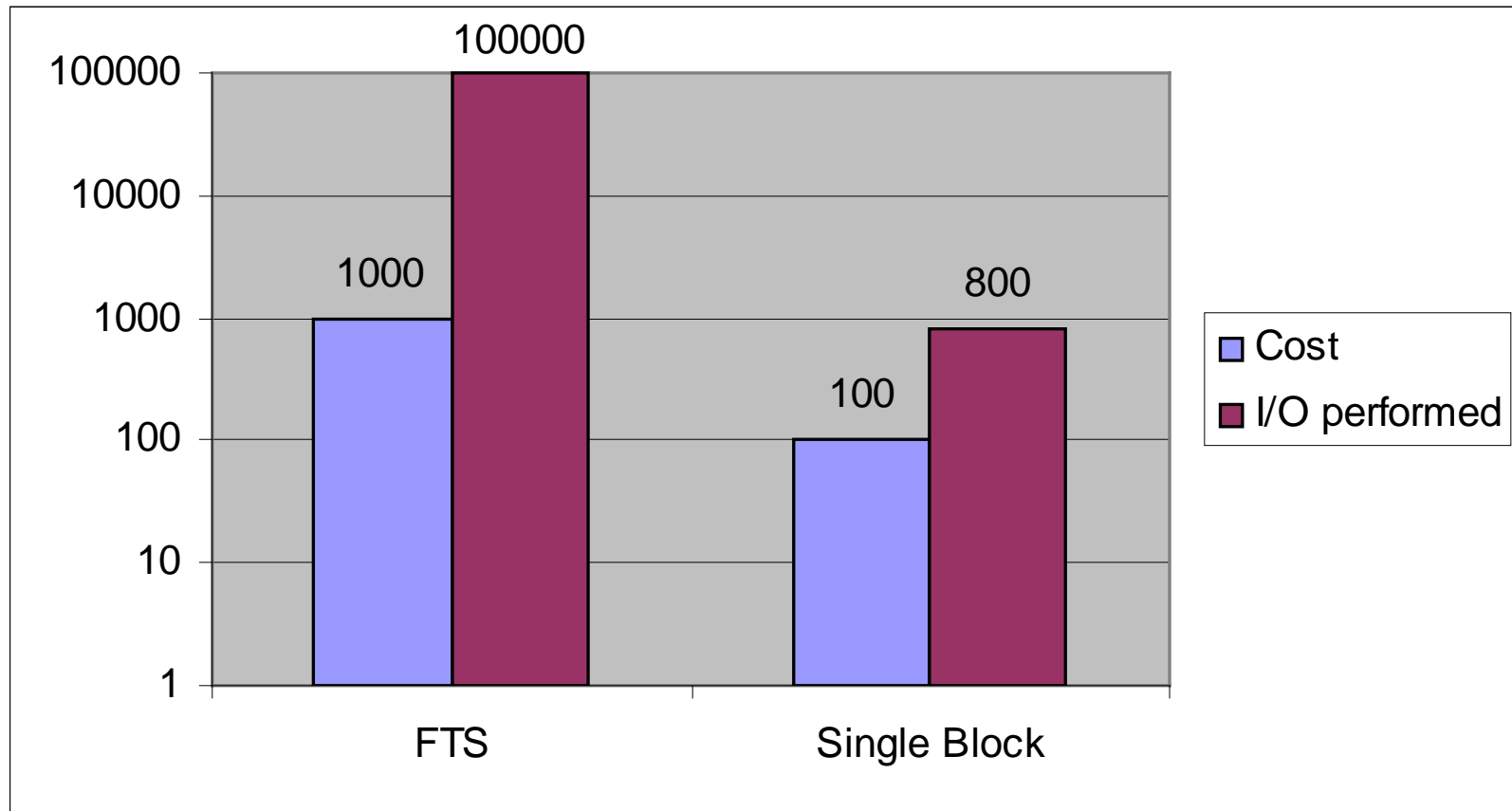
# Evolution of the CBO System Statistics

---

- Cost Full Table Scan with System Statistics:  
**1,000**
- Cost Full Table Scan with traditional costing:  
**100**
- This simplified example demonstrates that the cost calculated for Full Table Scan is potentially higher compared to traditional I/O based costing
- This tends to favor index access compared to traditional I/O based costing

# Evolution of the CBO System Statistics

---



# Evolution of the CBO System Statistics

---

- This addresses (partially) the issues that previously have been attempted to alleviate using the OICA and OIC parameters
- Furthermore the System Statistics introduce a CPU component
- Every operation (e.g. reading a block, locating a row/column in a block, applying functions or conversions to data) has a specific number of required CPU cycles assigned

# Evolution of the CBO System Statistics

---

- These number of CPU cycles can be again turned into a time and therefore cost since the System Statistics cover also the CPU speed
- This CPU cost will be added to the I/O cost and results in the final overall cost of an operation

# Evolution of the CBO System Statistics

---

- The CPU cost allows the optimizer to consider re-ordering the predicate evaluation
  - Instead of apply a costly conversion function 100,000 times and afterwards apply 1,000 simple comparisons on the data passing the first comparison, it might be less CPU intensive to apply the simple comparison 100,000 times and afterwards require the conversion only 10,000 times on the data passing the first comparison

# Evolution of the CBO System Statistics

---

```
create table predicate_order
as
select
    *
from
    all_objects
where
    rownum <= 1000;

-- gather statistics
exec dbms_stats.gather_table_stats(null, 'predicate_order', -
    method_opt=>'for all columns size 1')
```

# Evolution of the CBO System Statistics

---

```
explain plan for
select /*+ nocpu_costing */
      *
from
      predicate_order
where
      to_number(to_char(object_id)) = 5
and
      created > date '2008-01-01';
```

```
-----
| Id | Operation                | Name                |
-----
|  0 | SELECT STATEMENT         |                     |
|*  1 | TABLE ACCESS FULL      | PREDICATE_ORDER     |
-----
```

CPU costing disabled

Predicate Information (identified by operation id):

```
-----
1 - filter(TO_NUMBER(TO_CHAR("OBJECT_ID"))=5 AND "CREATED">TO_DATE
('2008-01-01 00:00:00', 'syyy-mm-dd hh24:mi:ss'))
```

# Evolution of the CBO System Statistics

---

```
explain plan for
select
    *
from
    predicate_order
where
    to_number(to_char(object_id)) = 5
and
    created > date '2008-01-01';
```

```
-----
| Id | Operation          | Name          |
-----
|  0 | SELECT STATEMENT   |               |
|*  1 | TABLE ACCESS FULL| PREDICATE_ORDER |
-----
```

CPU cost: 405702

Predicate Information (identified by operation id):

```
-----
1 - filter("CREATED">TO_DATE(' 2008-01-01 00:00:00', 'syyyy-mm-dd
        hh24:mi:ss') AND TO_NUMBER(TO_CHAR("OBJECT_ID"))=5)
```

# Evolution of the CBO System Statistics

---

```
explain plan for
select /*+ ordered_predicates */
      *
from
      predicate_order
where
      to_number(to_char(object_id)) = 5
and
      created > date '2008-01-01';
```

```
-----
| Id | Operation                | Name                |
-----
|  0 | SELECT STATEMENT         |                     |
|*  1 | TABLE ACCESS FULL      | PREDICATE_ORDER    |
-----
```

CPU cost: 605627

Predicate Information (identified by operation id):

```
-----
1 - filter(TO_NUMBER(TO_CHAR("OBJECT_ID"))=5 AND "CREATED">TO_DATE
('2008-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

# Evolution of the CBO System Statistics

---

- Extensible Optimizer: You can use the Oracle extensible optimizer to provide accurate information about your user-defined PL/SQL and table functions in terms of cardinality, selectivity and costs
- So if you're using user-defined PL/SQL in your SQL as part of predicates, please read e.g. the articles by OakTable fellow Adrian Billington on <http://www.oracle-developer.net>

# Evolution of the CBO System Statistics

---

- Upgrade Headaches: This predicate evaluation re-ordering can break an existing application if a statement relied so far on the predicate evaluation order, e.g. implicit TO\_NUMBER conversion on data with non-numeric content
- Different evaluation order can now throw „Invalid number“ errors because the conversion is applied to data that has been filtered out in the past by previous comparison steps

# Evolution of the CBO System Statistics

---

- Oracle deemed System Statistics so important that they decided to enable them ***by default*** from 10g on
- In Oracle 9i they were available, but had to be activated explicitly
- This caused a lot of headaches when upgrading from 9i to 10g, but in the long run a wise decision

# Understanding System Statistics

---

Different modes of System  
Statistics

# Different modes of System Statistics

---

- Cost = (  
#SRds \* SREADTIM +  
#MRds \* MREADTIM +  
#CPUCycles / CPUSPEED  
) / SREADTIM
  - #SRds - number of single block reads
  - #MRds - number of multi block reads
  - #CPUCycles - number of CPU Cycles
  - SREADTIM - single block read time
  - MREADTIM - multi block read time
  - CPUSPEED - CPU cycles per second

# Different modes of System Statistics

---

- Oracle supports different modes of System Statistics
- Two general approaches
  - **NOWORKLOAD** System Statistics
    - These are based on a limited set of „known“ basic values, e.g. the disk seek time, the disk transfer rate etc.
  - **WORKLOAD** System Statistics
    - These are based on an extended set of known values, e.g. the actual measured single block read time, multi block read time, the number of blocks actually read by a multi block read request etc. according to the workload performed

# Different modes of System Statistics

---

- Philosophy **NOWORKLOAD** System Statistics:
  - The actual "workload" of the system is unknown, therefore the calculations are based on the "raw" performance figures of the system capabilities like CPU speed, average I/O seek time and I/O transfer speed. Further information required for calculations will be derived from these basic figures

# Different modes of System Statistics - NOWORKLOAD

---

- Mechanics of the NOWORKLOAD System Statistics
  - Calculation is based on the following values (not modifiable in Oracle 9i):
    - CPUSPEEDNW: The CPU speed expressed in „standard Oracle CPU instructions per second“
    - IOSEEKTIM: The disk seek time in ms, default 10ms
    - IOTFRSPEED: The disk transfer speed in bytes per ms, default 4096 bytes/ms (approx. 4 MB/s)

# Different modes of System Statistics - NOWORKLOAD

---

- For the actual cost calculation we need however the time a single and a multi-block read request takes
- With NOWORKLOAD System Statistics these values are synthesized using the following formulas:
  - $SREADTIM = IOSEEKTIM + db\_block\_size / IOTFRSPEED$
  - $MREADTIM = IOSEEKTIM + mbrc * db\_block\_size / IOTFRSPEED$

# Different modes of System Statistics

---

- Philosophy **WORKLOAD** System Statistics:
  - An actual workload gets measured. From the workload statistics all values required for calculation are directly derived - the single- and multi-block read time, the average number of blocks read per multi-block read request and further information regarding parallel execution

# Different modes of System Statistics

---

- Measurement dilemma
  - The purpose of measuring WORKLOAD System Statistics is to base the calculations of the CBO on figures that represent the system capabilities. The resulting execution plans should take into account the system characteristics - eventually its goal is to improve performance by superior execution plans

# Different modes of System Statistics

---

- Measurement dilemma
  - But measuring System Statistics on a bad behaving system (e.g. far too many full table scans) will influence the measurement - depending on the underlying hardware the measured values might be biased towards full table scans, for instance very fast multi-block reads due to storage read-ahead and caching effects

# Different modes of System Statistics

---

- Measurement dilemma
  - No simple answer
  - One potential approach: Measure System Statistics on a regular basis into an user-defined statistics table - derive average values and set them manually
  - See for example Christian Antognini's "Troubleshooting Oracle Performance" for more information

# Different modes of System Statistics

---

- Oracle 9i introduced WORKLOAD System Statistics and default NOWORKLOAD System Statistics
- Both need to be enabled explicitly in Oracle 9i using  
`DBMS_STATS.GATHER_SYSTEM_STATS`  
and/or  
`DBMS_STATS.SET_SYSTEM_STATS`

# Different modes of System Statistics

---

- Oracle 10g extended this to three different modes:
- WORKLOAD System Statistics
- Default NOWORKLOAD System Statistics, which are **enabled by default** now from 10g on
- Gathered NOWORKLOAD System Statistics

# Different modes of System Statistics

---

- Important to understand:
  - All modes of System Statistics will result in the same final cost calculation formula
  - But the values used for the calculation, e.g. the time it takes to read a single/multi-block will be either be based on defaults or measures and potentially derived depending on the mode

# Understanding System Statistics

---

## Maintaining System Statistics

# Different modes of System Statistics - Maintenance

---

- DBMS\_STATS offers the following API for maintaining and modifying the System Statistics:

- Using

**DBMS\_STATS.GET\_SYSTEM\_STATS /  
SET\_SYSTEM\_STATS**

you can write your own set of System Statistics for both NOWORKLOAD and WORKLOAD values (NOWORKLOAD only 10g and later).

# Different modes of System Statistics - Maintenance

---

- You can use `DBMS_STATS.DELETE_SYSTEM_STATS` to remove the System Statistics, which will activate the default NOWORKLOAD System Statistics in 10g and disable CPU costing/System Statistics in 9i.

# Different modes of System Statistics - Maintenance

---

- You can use  
`DBMS_STATS.EXPORT_SYSTEM_STATS /`  
`IMPORT_SYSTEM_STATS`  
to export and import System Statistics  
to a user statistics table created with  
`DBMS_STATS.CREATE_STAT_TABLE`

# Different modes of System Statistics - Maintenance

---

- Note that **DBMS\_STATS.GATHER\_SYSTEM\_STATS** when used with an user stats table (created with **DBMS\_STATS.CREATE\_STAT\_TABLE**) behaves differently than e.g. **DBMS\_STATS.GATHER\_TABLE\_STATS**

# Different modes of System Statistics - Maintenance

---

- Whereas object related statistics always go to the data dictionary and you only have the option to save the current statistics to the user stats table before replacing them with the new values, **GATHER\_SYSTEM\_STATS** actually writes the System Statistics into the user stats table and doesn't change the actual System Statistics if you're supplying a user stats table name.

# Understanding System Statistics

---

## Multiple Block Sizes

# Different modes of System Statistics - Blocksizes

---

## Using multiple blocksizes

- What happens if you use tablespaces with different (non-default) block sizes?
- Let's first check what happens at actual execution time
- Using a MBRC runtime setting of 8 with 8KB default block size

# Different modes of System Statistics - Blocksizes

---

- The 8KB FTS does these multi-block reads, see the 10046 extended SQL trace with WAITs enabled:

```
WAIT #35: nam='db file scattered read' ela= 2296 file#=8  
  block#=12050 blocks=8 obj#=61857 tim=107988983808  
WAIT #35: nam='db file scattered read' ela= 916 file#=8  
  block#=12058 blocks=8 obj#=61857 tim=107988984970  
WAIT #35: nam='db file scattered read' ela= 911 file#=8  
  block#=12066 blocks=8 obj#=61857 tim=107988986104
```

# Different modes of System Statistics - Blocksizes

---

- The 2KB FTS does these multi-block reads, see the 10046 extended SQL trace with WAITs enabled:

```
WAIT #37: nam='db file scattered read' ela= 798 file#=6  
  block#=66  blocks=32 obj#=61859 tim=107991673671  
WAIT #37: nam='db file scattered read' ela= 828 file#=6  
  block#=98  blocks=32 obj#=61859 tim=107991674822  
WAIT #37: nam='db file scattered read' ela= 829 file#=6  
  block#=130 blocks=32 obj#=61859 tim=107991675980
```

# Different modes of System Statistics - Blocksizes

---

- So the runtime engine scales the multi-block read accordingly, and there is no difference in the size of the actual read request
- Due to this you'll notice that the runtime performance difference is usually negligible

# Different modes of System Statistics - Blocksizes

---

- Only difference is the different number of consistent gets when using buffered gets (db file scattered read), which might lead to different latch and CPU activity
- Usually the largest contributor to execution time is however the wait on the actual I/O, so the latch/CPU part is not significant in most cases

# Different modes of System Statistics - Blocksizes

---

## Using multiple blocksizes

- Let's check the optimizer's cost calculations
- First the traditional I/O costing
- Remember that we've seen that at runtime the actual multi-block I/O size will be the same (scaled accordingly)

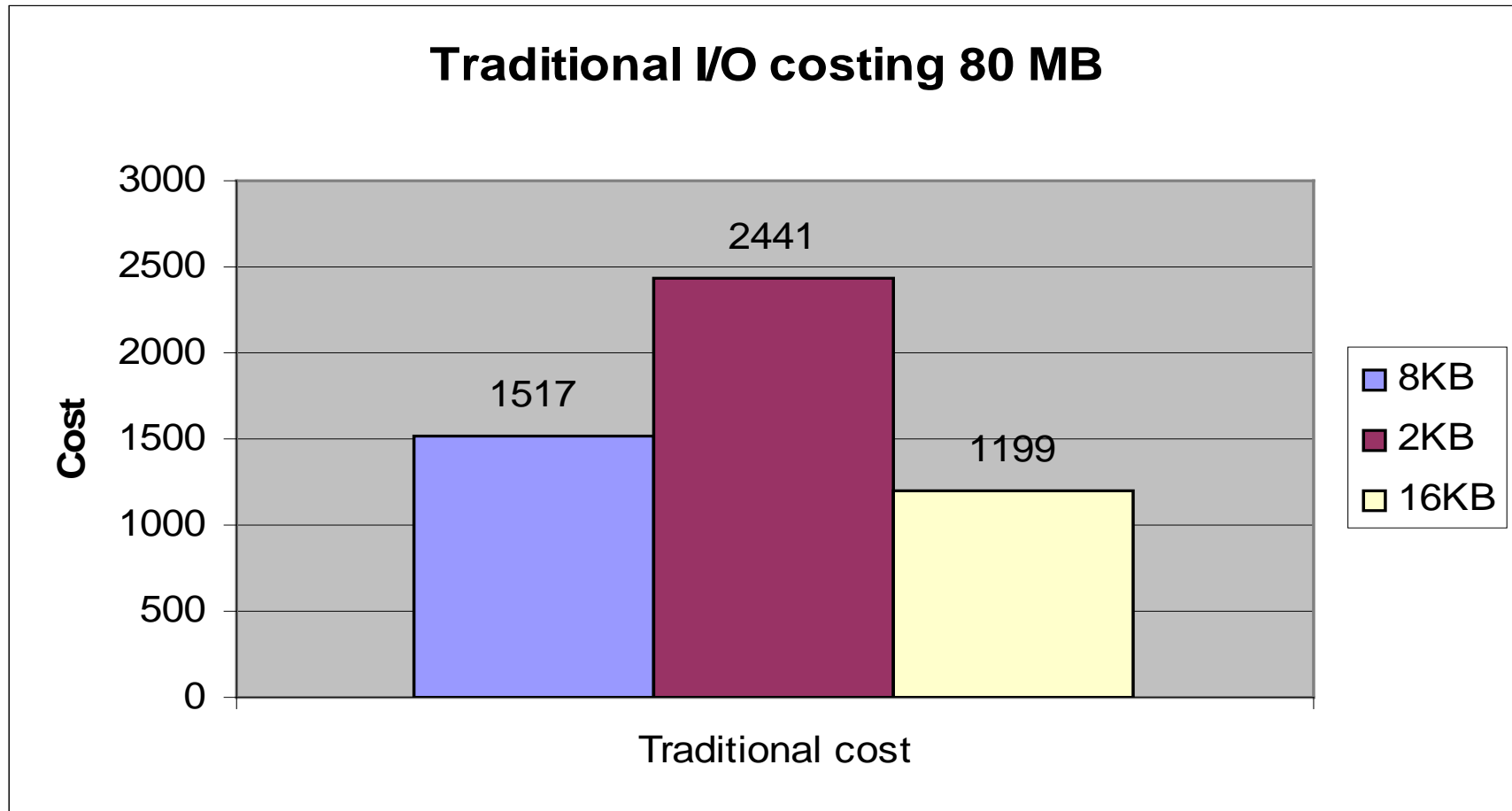
# Different modes of System Statistics - Blocksizes

---

- Testcase uses a 10,000 blocks / 1 row per block table on 10.2.0.4 in a Manual Segment Space Managed (MSSM) Locally Managed Tablespace (LMT) with default 8 KB blocksize

# Different modes of System Statistics - Blocksizes

---



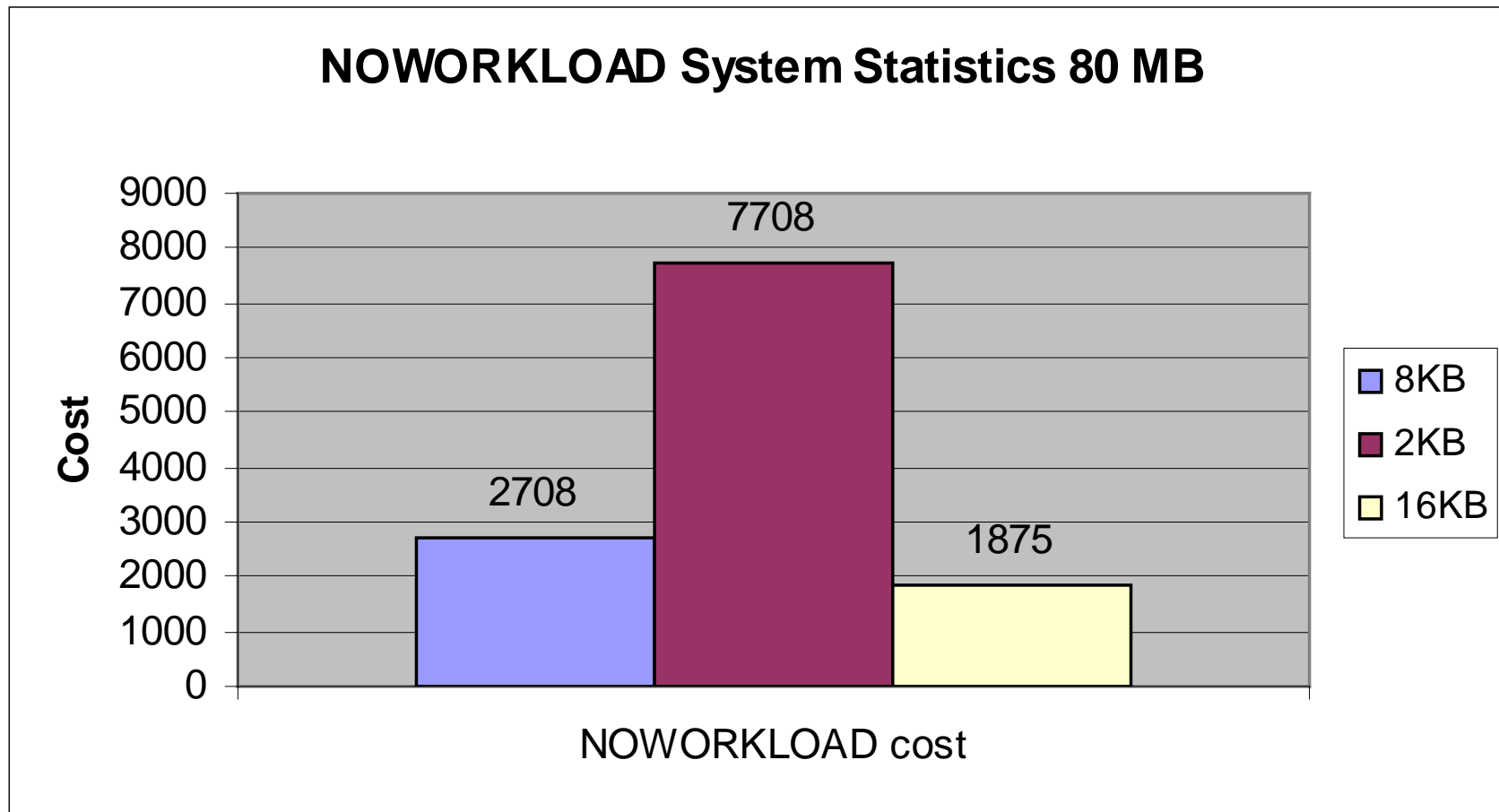
# Different modes of System Statistics - Blocksizes

---

- Now the NOWORKLOAD System Statistics
- Remember again that we've seen that at runtime the actual multi-block I/O size will be the same (scaled accordingly)

# Different modes of System Statistics - Blocksizes

---



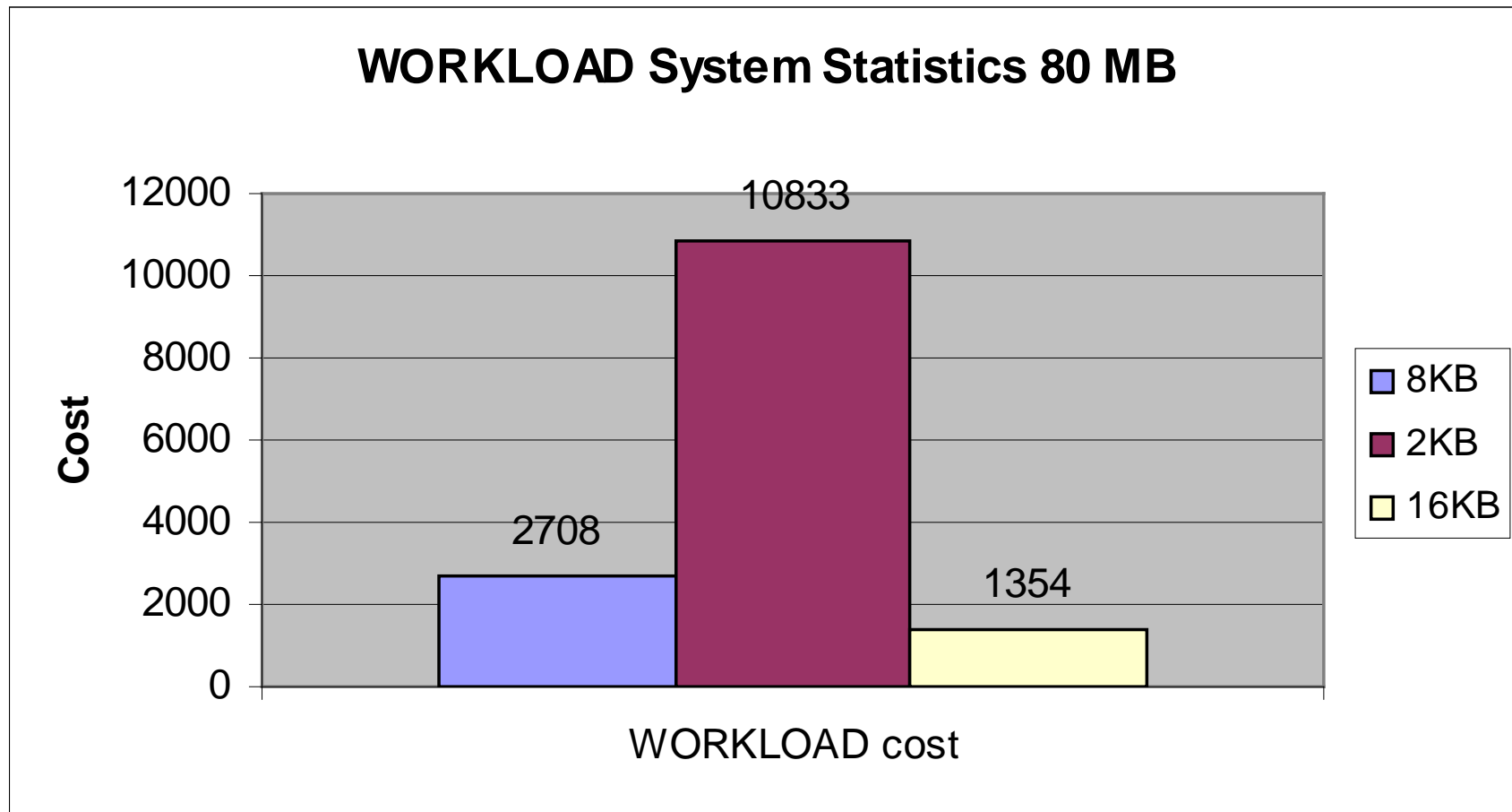
# Different modes of System Statistics - Blocksizes

---

- Finally the WORKLOAD System Statistics
- Remember again that we've seen that at runtime the actual multi-block I/O size will be the same (scaled accordingly)

# Different modes of System Statistics - Blocksizes

---



# Different modes of System Statistics - Blocksizes

---

- Each of the different modes has its specific deficiencies when dealing with objects in non-default blocksizes
- The somehow odd thing is that the traditional I/O costing does the best job, and all System Statistics based calculations are utterly wrong

# Different modes of System Statistics - Blocksizes

---

## **Traditional I/O based costing**

- The traditional I/O based costing simply scales the MBRC up or down according to the non-default blocksize to come to the same I/O read request size

# Different modes of System Statistics - Blocksizes

---

- So if you e.g. have a MBRC of 8 and a default blocksize of 8KB and now calculate the cost for an object residing in a 2KB tablespace, the MBRC will be multiplied 4, which results in a MBRC of 32.

# Different modes of System Statistics - Blocksizes

---

- The I/O cost will be different although due to the different adjustment used with the higher MBRC setting
- The *adjusted* MBRC for 32 is 16.39 whereas the adjusted MBRC for 8 is 6.59, so the calculated cost for the full table scan of the object residing in the 2KB tablespace will be higher.

# Different modes of System Statistics - Blocksizes

---

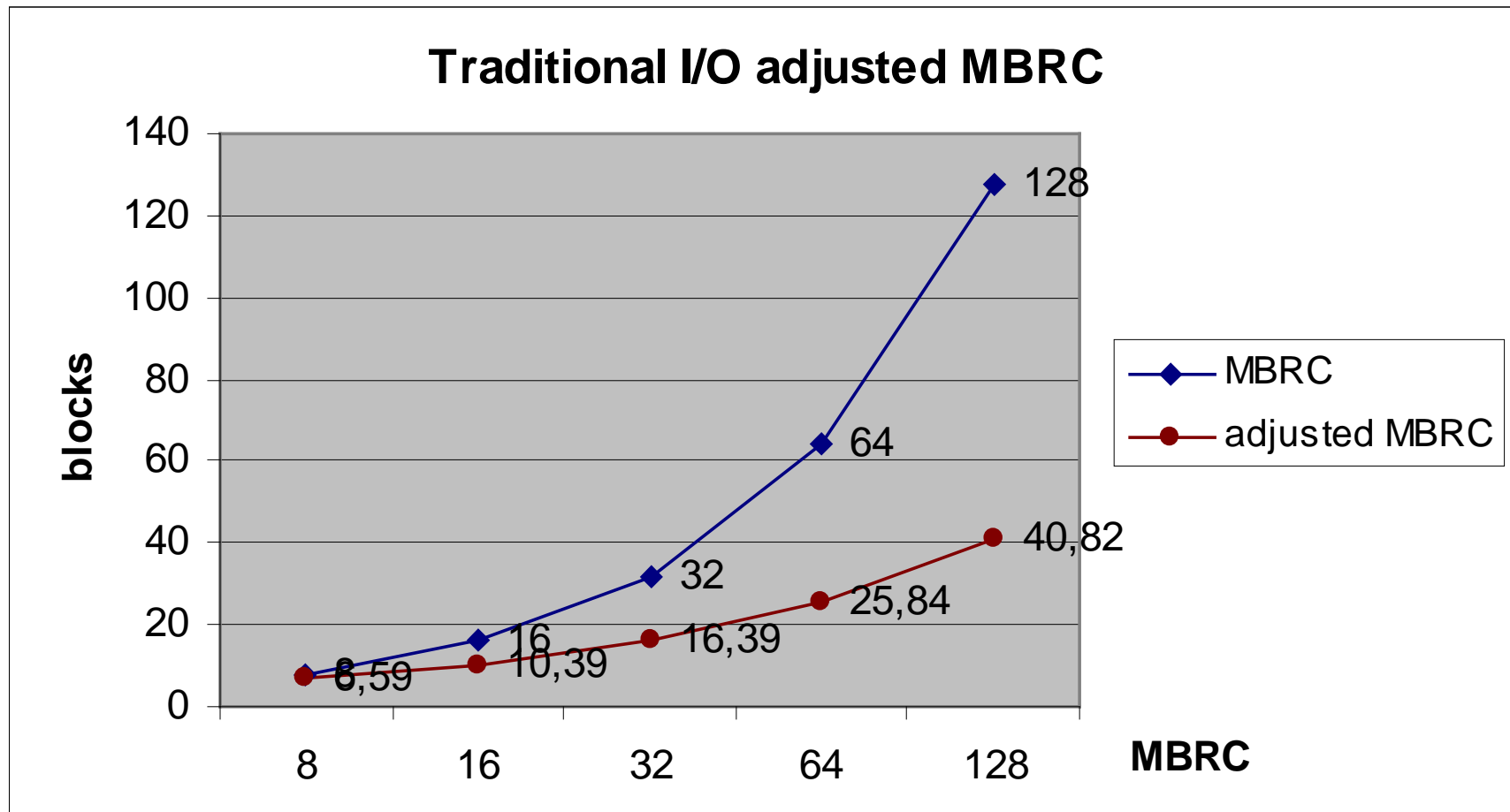
- Likewise the same happens when using an object in a 16KB non-default tablespace. The MBRC will be reduced accordingly to 4 to get the same I/O read size again. Since adjusted MBRC for MBRC = 4 is 4.17, the cost calculated will actually be less for the object residing the 16KB tablespace.

# Different modes of System Statistics - Blocksizes

---

- With Traditional Costing the cost of the Full Table Scan will simply be the number of blocks to read divided by the "adjusted" MBRC

# Different modes of System Statistics-NOWORKLOAD



# Different modes of System Statistics - Blocksizes

---

## System Statistics

- Things get more complicated when using NOWORKLOAD or WORKLOAD System Statistics.
- The formula to calculate the I/O cost of a full table scan with System Statistics is:

Number of blocks / MBRC \* MREADTIM / SREADTIM

# Different modes of System Statistics - Blocksizes

---

- And in case of NOWORKLOAD System Statistics the MREADTIM and SREADTIM are synthesized using the following formula:
- $SREADTIM = IOSEEKTIM + DB\_BLOCK\_SIZE / IOTRFSPEED$
- $MREADTIM = IOSEEKTIM + DB\_BLOCK\_SIZE * MBRC / IOTRFSPEED$

# Different modes of System Statistics - Blocksizes

---

- Now if the object resides in a non-default blocksize tablespace, the following inconsistent adjustments are applied to the formulas

# Different modes of System Statistics - Blocksizes

---

## **NOWORKLOAD System Statistics**

- SREADTIM as above, using ***DEFAULT*** DB\_BLOCK\_SIZE
- MREADTIM = IOSEEKTIM + ***DEFAULT*** DB\_BLOCK\_SIZE \* ***scaled MBRC*** / IOTRFSPEED
- I/O cost = Number of blocks / scaled MBRC \* MREADTIM / SREADTIM

# Different modes of System Statistics - Blocksizes

---

- So obviously something is odd in above formulas: The SREADTIM and MREADTIM values are synthesized with a mixture of a scaled MBRC (according to the block size) but a non-adjusted default DB\_BLOCK\_SIZE, resulting in a large variation in cost.

# Different modes of System Statistics - Blocksizes

---

- So a full table scan in a small blocksize is calculated to be much more expensive than in the default blocksize, and likewise a full table scan in a large blocksize is much cheaper

# Different modes of System Statistics - Blocksizes

---

- Unfortunately this doesn't reflect at all the runtime behaviour, since Oracle actually scales the I/O read request size accordingly meaning that the runtime difference usually is negligible, but the cost calculated is dramatically different.

# Different modes of System Statistics - Blocksizes

---

## WORKLOAD System Statistics

- MBRC as measured/set
- SREADTIM as measured/set
- MREADTIM as measured/set
  
- I/O cost = Number of blocks /  
***(unadjusted)*** MBRC \* MREADTIM /  
SREADTIM

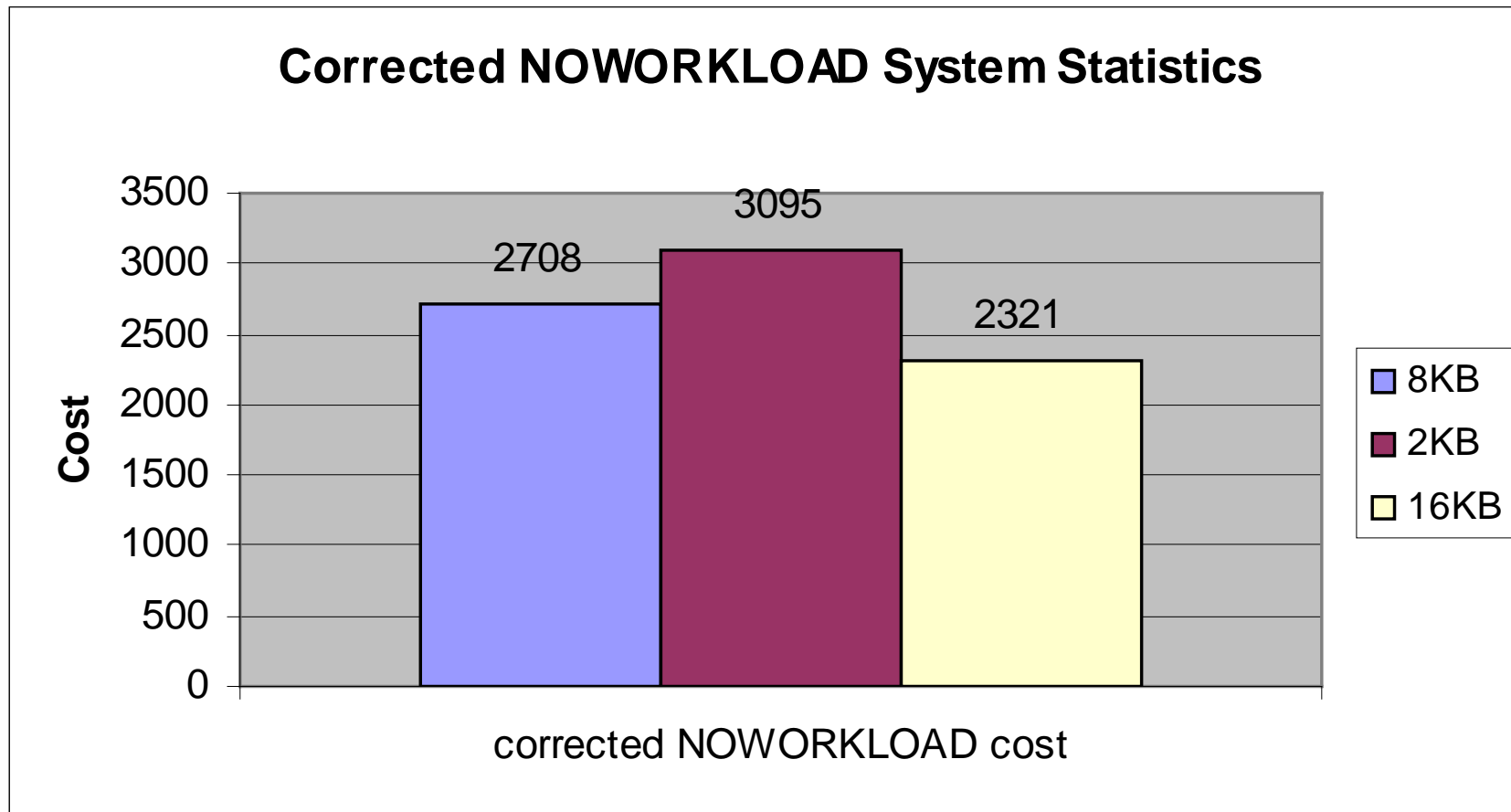
# Different modes of System Statistics - Blocksizes

---

- This is even worse than above NOWORKLOAD result because the I/O cost calculated simply is different by the factor of number of blocks in non-default block size / number of blocks in default block size, e.g. an object residing in a 2KB block size will have an I/O cost four times higher than an object residing in a 8KB default blocksize, and the MBRC is not adjusted at all for the calculation.

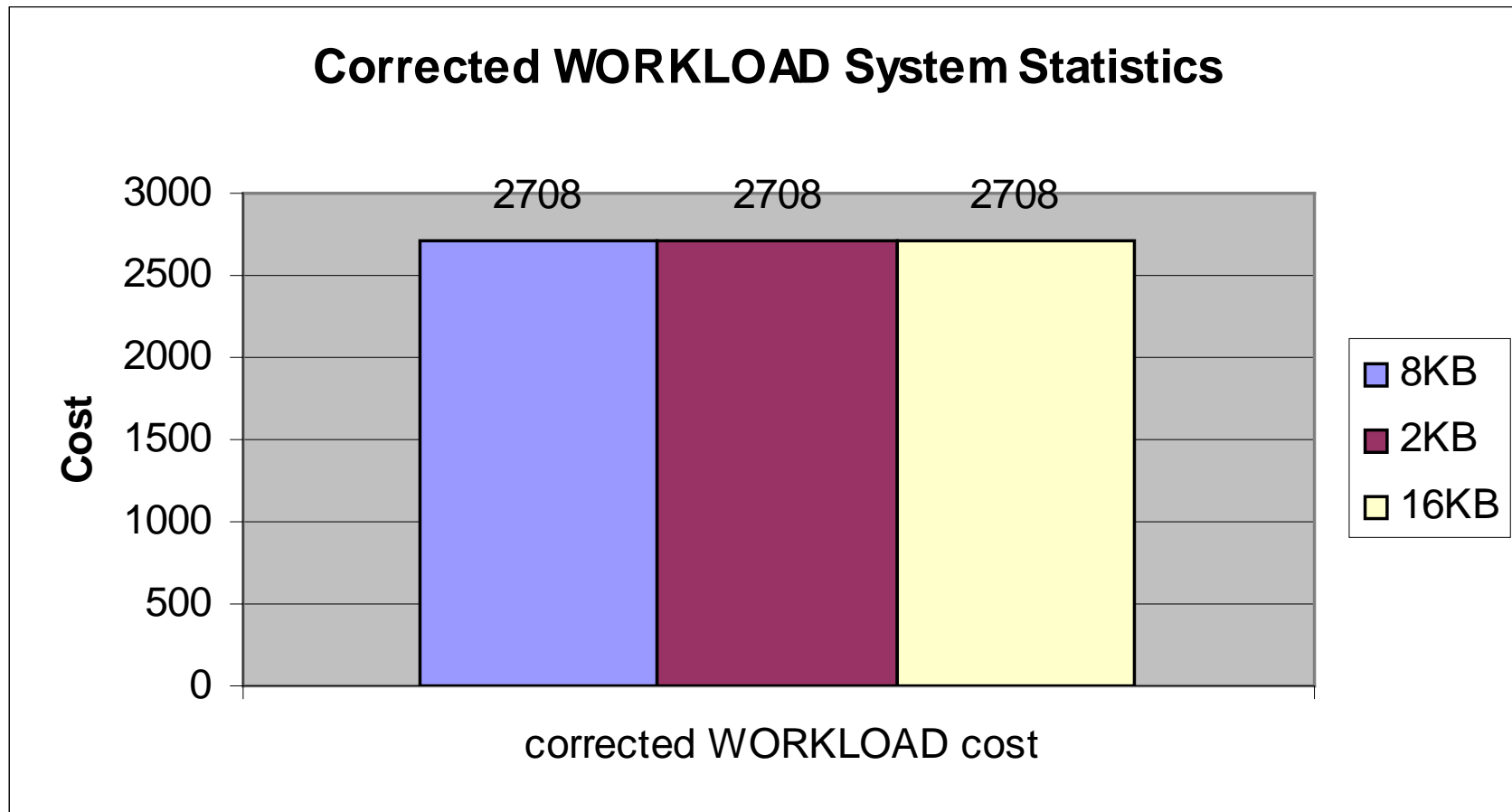
# Different modes of System Statistics - Blocksizes

---



# Different modes of System Statistics - Blocksizes

---



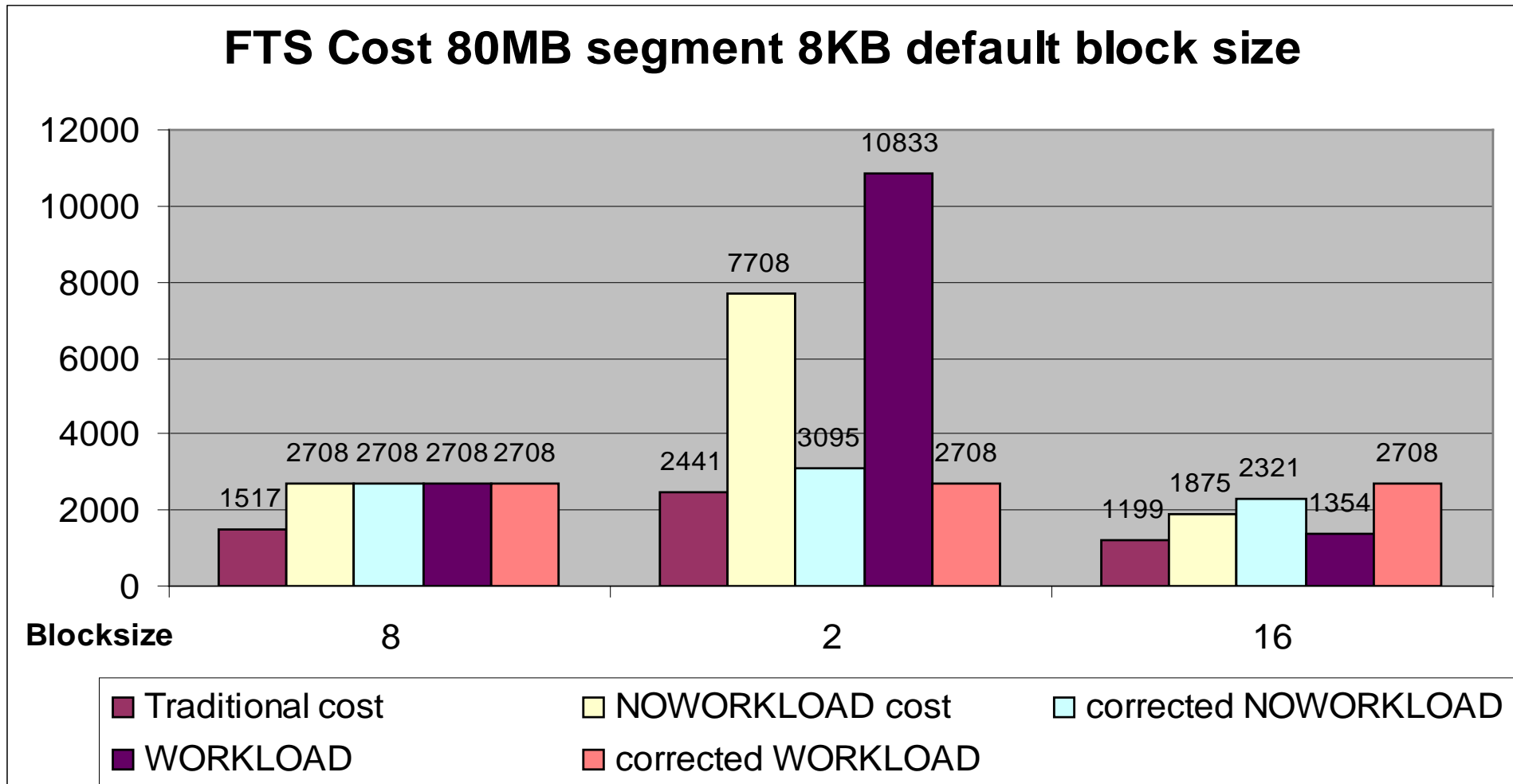
# Different modes of System Statistics - Blocksizes

---

## Using multiple blocksizes

- The cost based optimizer does a bad job when it comes to calculating costs for full table scans of objects residing in non-default block sizes. It looks like that this feature has been introduced to support transportable tablespaces but it shows some shortcomings when it comes to cost calculation.

# Different modes of System Statistics - Blocksizes



# Different modes of System Statistics - Summary

---

- System Statistics have been introduced in Oracle 9i to address the shortcomings of the traditional I/O based costing
- Different modes
  - NOWORKLOAD
  - WORKLOAD
- Enabled by default from 10g on using default NOWORKLOAD mode

# Different modes of System Statistics - Summary

---

- Ideally based on measured WORKLOAD System Statistics that **properly** reflect the capabilities of the underlying system/hardware
- The most significant changes are the increased cost of FTS/IFFS operations and the potential predicate re-ordering due to CPU costs considered

# Different modes of System Statistics - Summary

---

- Don't believe everything the "Internet" suggests; in particular don't use different block sizes for tuning purposes if FTS/IFFS operations are involved
- The cost calculations in these cases will not reflect the actual runtime behaviour
- Use different block sizes when:
  - using transportable Tablespaces
  - using LOBs for the LOB segment

# Different modes of System Statistics - Q & A

---

Questions

&

Answers

# Different modes of System Statistics - References

---

- “Cost Based Oracle - Fundamentals” by Jonathan Lewis
- Metalink Note 457228.1: How To Calculate CPU Cost
- Oracle9i Database Performance Tuning Guide and Reference: Release 2 (9.2) Part Number A96533-02
- <http://www.oracle-developer.net> by Adrian Billington

# Different modes of System Statistics - The End

---

Thank you!

# Understanding System Statistics

---

The gory details

# Understanding System Statistics

---

## NOWORKLOAD System Statistics

# Different modes of System Statistics - NOWORKLOAD

---

- Mechanics of the NOWORKLOAD System Statistics
  - Calculation is based on the following values (not modifiable in Oracle 9i):
    - CPUSPEEDNW: The CPU speed expressed in „standard Oracle CPU instructions per second“
    - IOSEEKTIM: The disk seek time in ms, default 10ms
    - IOTFRSPEED: The disk transfer speed in bytes per ms, default 4096 bytes/ms (approx. 4 MB/s)

# Different modes of System Statistics - NOWORKLOAD

---

- For the actual cost calculation we need however the time a single and a multi-block read request takes
- With NOWORKLOAD System Statistics these values are synthesized using the following formulas:
  - $SREADTIM = IOSEEKTIM + db\_block\_size / IOTFRSPEED$
  - $MREADTIM = IOSEEKTIM + mbrc * db\_block\_size / IOTFRSPEED$

# Different modes of System Statistics - NOWORKLOAD

---

- $SREADTIM = IOSEEKTIM + db\_block\_size / IOTFRSPEED$
- In words expressed: The time to read a single block is the time it takes the disk to seek plus the time it takes to transfer a block expressed in milliseconds

# Different modes of System Statistics - NOWORKLOAD

---

- The IOSEEKTIM is expressed in milliseconds, default 10 ms
- The db\_block\_size is expressed in bytes, default value is database block size
- The IOTFRSPEED is expressed in bytes per millisecond, default value 4096 bytes per millisecond (4 MB / s)

# Different modes of System Statistics - NOWORKLOAD

---

- Example calculation with the default values and an assumed block size of 8KB
  - SREADTIM = 10 ms
  - + 8192 bytes / (4096 bytes / ms) =
  - 10 ms + 2 ms = 12 ms
- Using larger block sizes will result in a larger derived SREADTIM with the remaining values left unchanged

# Different modes of System Statistics - NOWORKLOAD

---

- $MREADTIM = IOSEEKTIM + mbrc * db\_block\_size / IOTFRSPEED$
- In words expressed: The time to perform a multi block read request is the time it takes the disk to seek plus the time it takes to transfer the number of blocks defined for a multi-block read expressed in milliseconds

# Different modes of System Statistics - NOWORKLOAD

---

- mbrc: This value represents the number of blocks a multi-block read request is supposed to read
- Its actual value depends on the version of Oracle and the instance configuration
- In 9i this corresponds to the “db\_file\_multiblock\_read\_count” setting, which defaults to 16 if left unset

# Different modes of System Statistics - NOWORKLOAD

---

- In 10.2 and later this corresponds to the “db\_file\_multiblock\_read\_count” setting if this is explicitly set
- However, if “db\_file\_multiblock\_read\_count” is left unset then two new internal parameters become significant:
  - “\_db\_file\_optimizer\_read\_count”
  - “\_db\_file\_exec\_read\_count”

# Different modes of System Statistics - NOWORKLOAD

---

- "\_db\_file\_optimizer\_read\_count"  
This defaults to 8 and will be used as "mbrc" in the MREADTIM formula
- "\_db\_file\_exec\_read\_count"  
This is platform specific and defaults to the largest possible I/O request, which is usually 1MB (128 blocks with 8KB block size). This will be used at runtime for the actual execution of the multi-block read request

# Different modes of System Statistics-NOWORKLOAD

---

- "\_db\_file\_optimizer\_read\_count", "\_db\_file\_exec\_read\_count":
- These two parameters introduced in 10.2 allow the database to use a conservative setting of "mbrc" for calculation purposes, but a much more aggressive setting at actual execution time

# Different modes of System Statistics-NOWORKLOAD

---

- Example calculation with the default values and an assumed block size of 8KB
  - $MREADTIM = 10 \text{ ms}$ 
    - +  $8 * 8192 \text{ bytes} / (4096 \text{ bytes} / \text{ms}) =$
    - $10 \text{ ms} + 65536 \text{ bytes} / (4096 \text{ bytes} / \text{ms}) =$
    - $10 \text{ ms} + 16 \text{ ms} = 26 \text{ ms}$
- Using larger block sizes will result in a larger derived MREADTIM with the remaining values left unchanged

# Different modes of System Statistics-NOWORKLOAD

---

- These synthesized values will then be slotted into the previously shown formula to calculate the cost
- Re-writing the formula by dividing by SREADTIM and omitting the CPU cost component we get the following:
- number of single-block reads +  
number of multi-block reads \*  
MREADTIM / SREADTIM

# Different modes of System Statistics-NOWORKLOAD

---

- Therefore we can derive that in comparison to traditional I/O based costing that simply counts the number of I/O requests the cost of a full table scan / fast full index scan operation will increase approximately by the factor  $MREADTIM/SREADTIM$

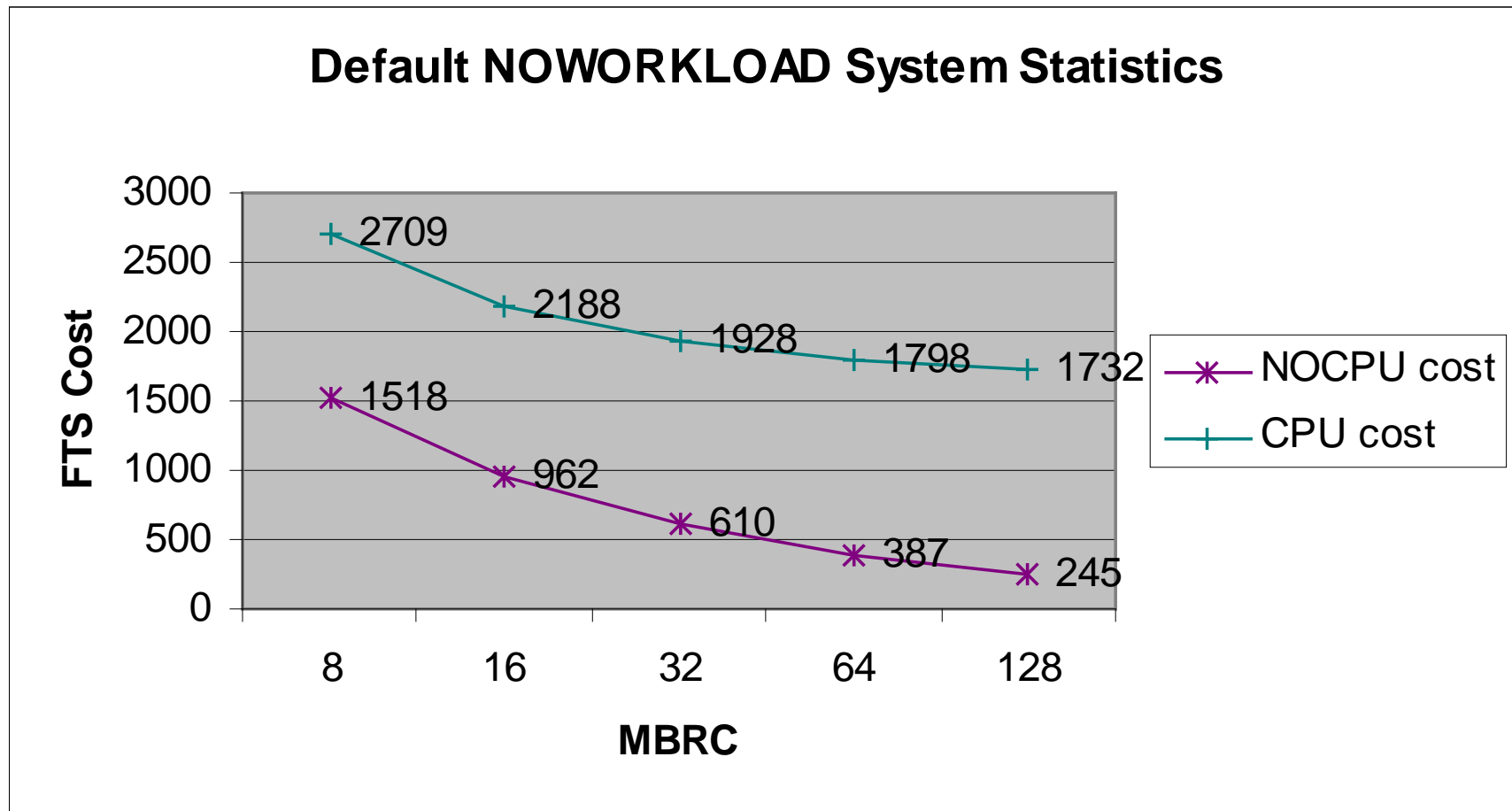
# Different modes of System Statistics-NOWORKLOAD

---

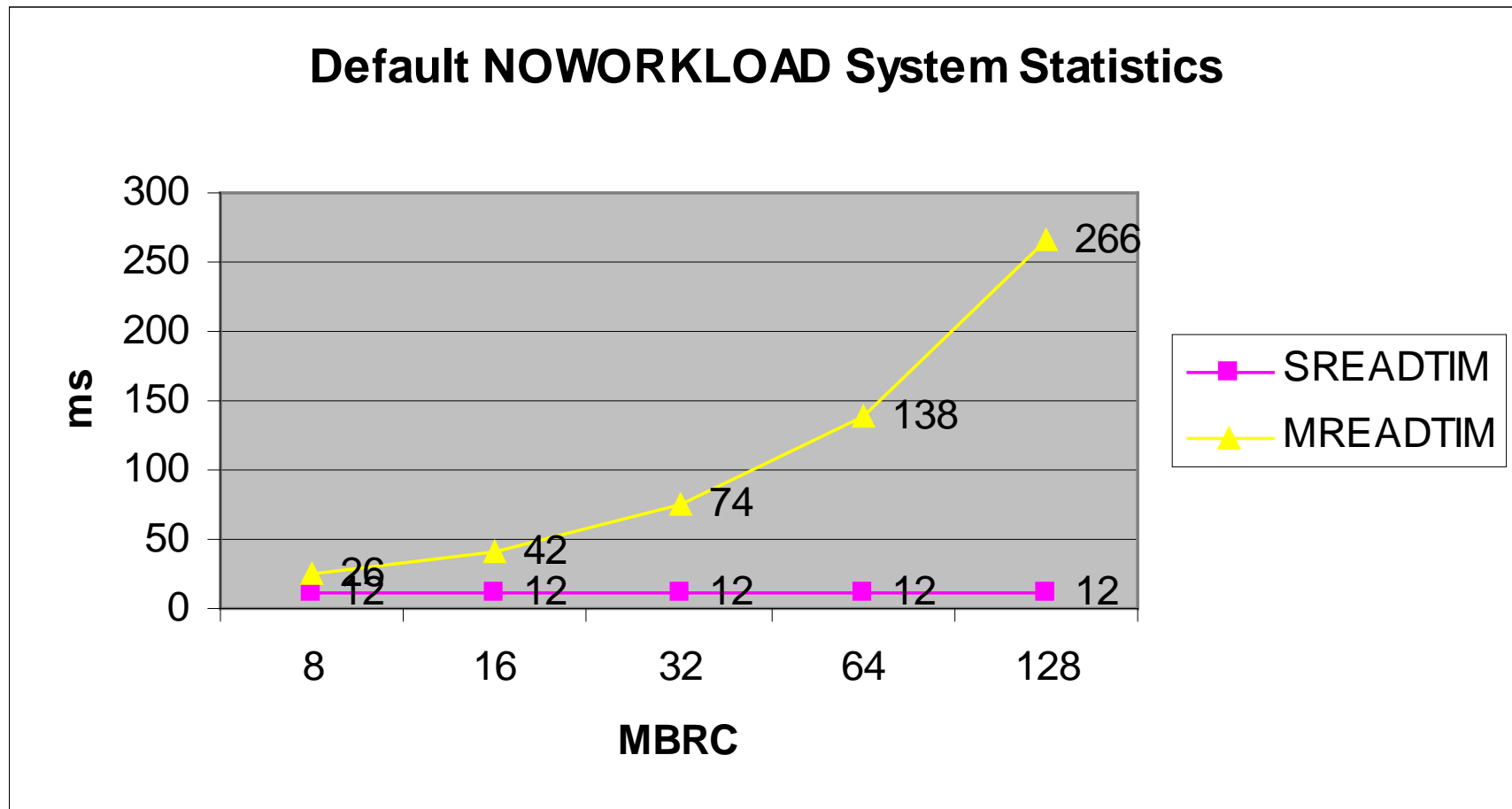
- Comparison of default NOWORKLOAD System Statistics with traditional I/O costing for a Full Table Scan operation
- Testcase uses a 10,000 blocks / 1 row per block table on 10.2.0.4 in a Manual Segment Space Managed (MSSM) Locally Managed Tablespace (LMT) with 8 KB blocksize

# Different modes of System Statistics-NOWORKLOAD

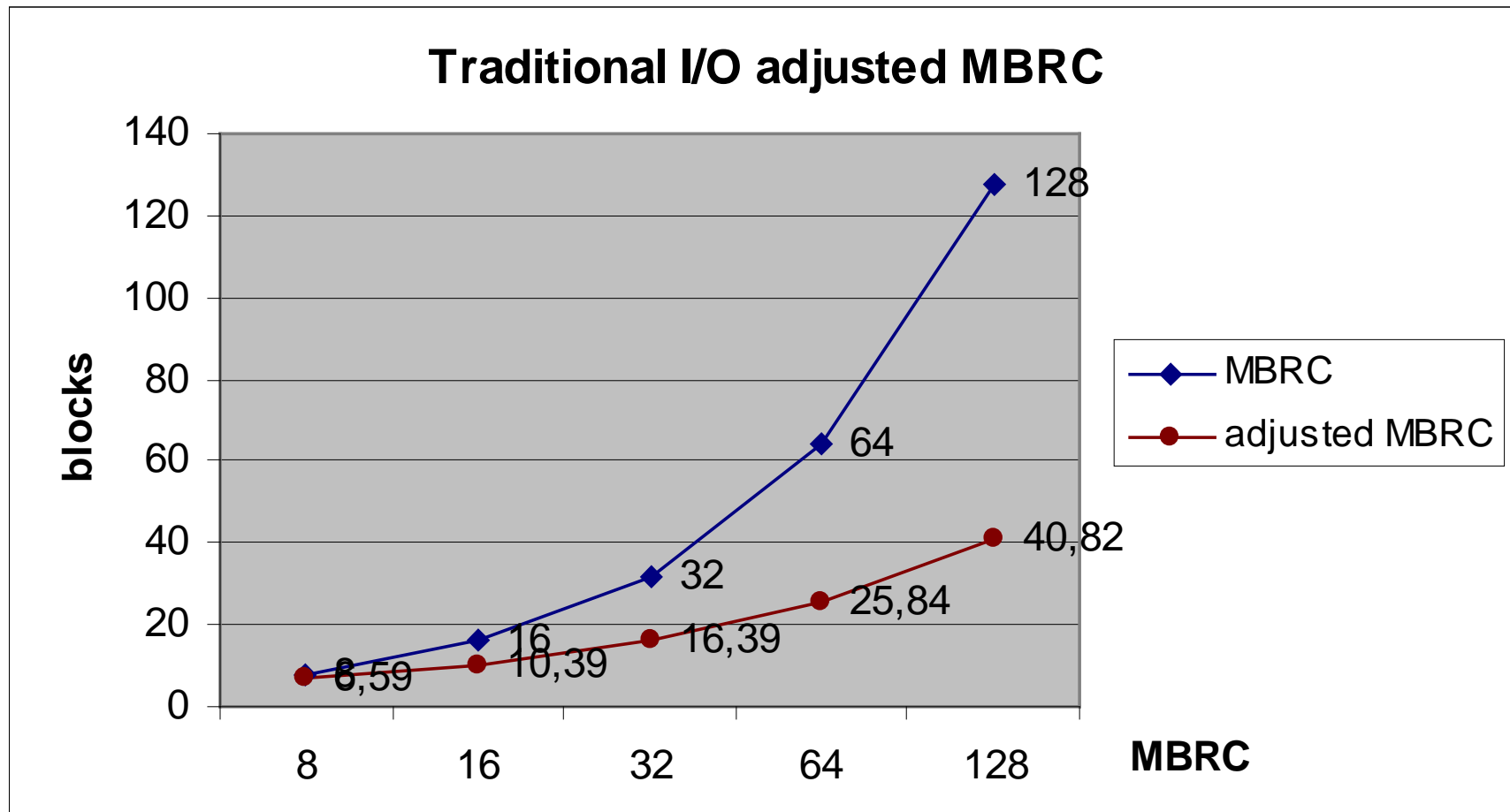
---



# Different modes of System Statistics-NOWORKLOAD



# Different modes of System Statistics-NOWORKLOAD



# Different modes of System Statistics-NOWORKLOAD

---

- Gathered NOWORKLOAD System Statistics:
  - In 10g and later you can gather the values used for the NOWORKLOAD calculation using  
DBM\_STATS.GATHER\_SYSTEM\_STATS  
with the parameter NOWORKLOAD:

```
DBMS_STATS.GATHER_SYSTEM_STATS( 'NOWORKLOAD' )
```

# Different modes of System Statistics-NOWORKLOAD

---

- This actually gathers the IOTFRSPEED and IOSEEKTIM values in addition to CPUSPEEDNW rather than using the default values of 4096 and 10.
- Puts artificial load on your system by running random I/O on your datafiles

# Different modes of System Statistics-NOWORKLOAD

---

- Repeating the testcase with the 10,000 blocks segment with gathered NOWORKLOAD System Statistics shows that the calculated cost is significantly different
- Let's first review what NOWORKLOAD System Statistics have been gathered on the test system

# Different modes of System Statistics-NOWORKLOAD

```
SQL> column sname format a20
SQL> column pname format a20
SQL> column pval2 format a20
SQL>
SQL> select
  2  sname, pname, pval1, pval2 from
  3  sys.aux_stats$;
```

SNAME	PNAME	PVAL1	PVAL2
SYSSTATS_INFO	STATUS		COMPLETED
SYSSTATS_INFO	DSTART		04-26-2009 14:21
SYSSTATS_INFO	DSTOP		04-26-2009 14:21
SYSSTATS_INFO	FLAGS	1	
SYSSTATS_MAIN	CPUSPEEDNW	1000000	
<b>SYSSTATS_MAIN</b>	<b>IOSEKTIM</b>	<b>14.226</b>	
<b>SYSSTATS_MAIN</b>	<b>IOTFRSPEED</b>	<b>32517.754</b>	
SYSSTATS_MAIN	SREADTIM		
SYSSTATS_MAIN	MREADTIM		
SYSSTATS_MAIN	CPUSPEED		
SYSSTATS_MAIN	MBRC		
SYSSTATS_MAIN	MAXTHR		
SYSSTATS_MAIN	SLAVETHR		

# Different modes of System Statistics-NOWORKLOAD

---

```
SQL> explain plan for
  2  select
  3      max(val)
  4  from
  5      t1;
```

```
-----
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	4	<b>1403</b> (0)	00:00:21
1	SORT AGGREGATE		1	4		
2	TABLE ACCESS FULL	T1	10000	40000	<b>1403</b> (0)	00:00:21

```
-----
```

# Understanding System Statistics

---

## WORKLOAD System Statistics

# Different modes of System Statistics - WORKLOAD

---

## Gathered WORKLOAD System Statistics

- Gathering WORKLOAD System Statistics measures a different set of values, including the actual MBRC, SREADTIM and MREADTIM values.

# Different modes of System Statistics - WORKLOAD

---

- The cost calculation therefore doesn't use the synthesized SREADTIM and MREADTIM values any longer, nor does it use the "\_db\_file\_optimizer\_read\_count" parameter for the MBRC in 10g and later, but uses simply the measured values.

# Different modes of System Statistics - WORKLOAD

---

- Therefore the I/O costs calculated with WORKLOAD System Statistics are not dependent on the "db\_file\_multiblock\_read\_count" value used, but the important point to keep in mind is that the gathered WORKLOAD System Statistics are based on the "db\_file\_multiblock\_read\_count" (in 10g and later on the internal parameter "\_db\_file\_exec\_read\_count")

# Different modes of System Statistics - WORKLOAD

---

- So the values measured are obviously influenced by this setting ("`_db_file_exec_read_count`" equals "`db_file_multiblock_read_count`" if this has been set and the underscore parameter hasn't been modified).
- Note that in 10g and later the runtime engine still uses the "`_db_file_exec_read_count`", regardless of the MBRC used to calculate the cost<sub>419</sub>

# Different modes of System Statistics - WORKLOAD

---

- Gathering WORKLOAD System Statistics, first option:

```
exec DBMS_STATS.GATHER_SYSTEM_STATS('START')  
-- some significant (ideally "representative")  
   workload needs to be performed  
-- otherwise some or all of the measured values  
   will be missing  
exec DBMS_STATS.GATHER_SYSTEM_STATS('STOP')
```

# Different modes of System Statistics - WORKLOAD

---

- Gathering WORKLOAD System Statistics, second option:

```
exec DBMS_STATS.GATHER_SYSTEM_STATS(  
  'INTERVAL',  
  <interval_in_minutes>)
```

# Different modes of System Statistics - WORKLOAD

---

```
SQL> column sname format a20
SQL> column pname format a20
SQL> column pval2 format a20
SQL>
SQL> select
2 sname, pname, pval1, pval2
3 from sys.aux_stats$;
```

SNAME	PNAME	PVAL1	PVAL2
SYSSTATS_INFO	STATUS		COMPLETED
SYSSTATS_INFO	DSTART		05-03-2009 13:24
SYSSTATS_INFO	DSTOP		05-03-2009 13:24
SYSSTATS_INFO	FLAGS		1
SYSSTATS_MAIN	CPUSPEEDNW	1000000	
SYSSTATS_MAIN	IOSEKTIM		10
SYSSTATS_MAIN	IOTFRSPEED		4096
<b>SYSSTATS_MAIN</b>	<b>SREADTIM</b>	<b>8.021</b>	
<b>SYSSTATS_MAIN</b>	<b>MREADTIM</b>	<b>12.93</b>	
<b>SYSSTATS_MAIN</b>	<b>CPUSPEED</b>	<b>1000000</b>	
<b>SYSSTATS_MAIN</b>	<b>MBRC</b>	<b>8</b>	
SYSSTATS_MAIN	MAXTHR		
SYSSTATS_MAIN	SLAVETHR		

# Different modes of System Statistics - WORKLOAD

---

```
SQL> explain plan for
 2  select
 3      max(val)
 4  from
 5      t1;
```

```
-----
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
0	SELECT STATEMENT		1	4	<b>2016</b> (0)	00:00:17	
1	SORT AGGREGATE		1	4			
2	TABLE ACCESS FULL	T1	10000	40000	<b>2016</b> (0)	00:00:17	

```
-----
```