CBO - A Configuration Roadmap

Christian Antognini
Trivadis AG
Zürich, Switzerland

Keywords

CBO, cost-based optimizer, initialization parameters, object statistics, system statistics, cost model, PGA management

Abstract

The cost-based optimizer (CBO) isn't only one of the most complex pieces of software that constitute the Oracle kernel, it's also one of the most unappreciated. Why? Because to effectively use the CBO you have to correctly configure it and, therefore, understand how it works! In fact, without an optimal configuration, the CBO will generate poor execution plans that lead to poor performance. Such a configuration consists not only of some initialization parameters (INIT.ORA parameters), it also depends on system and object statistics. This paper, while explaining how different initialization parameters and statistics influence the CBO, proposes a straightforward and pragmatic roadmap to achieve the wanted configuration in 9i and 10g.

Disclaimer

The CBO changes from release to release. Sometimes patches even enable/disable features! This situation makes it difficult to give any general advice that applies to different versions. Therefore, this paper focuses on the most important versions at present:
- Oracle9i Release 2 (9.2)
- Oracle Database 10g Release 1 (10.1)
- Oracle Database 10g Release 2 (10.2)

When information doesn't apply to all three versions, it will be explicitly written.

The formulas provided in this paper, with a single exception, aren't published by Oracle. Different tests show that they are able to describe how the CBO estimates the cost of a given operation. In any case they neither pretend to be precise nor correct in all situations. I provide them to give you an idea on how an initialization parameter or statistic influences the CBO estimations.

I am not able in a short paper to cover in detail all necessary areas. For this reason I assume that you know the Oracle architecture and have some basic knowledge of the CBO and the package DBMS_STATS. In other words I assume you have been working with Oracle over the last couple of years.

Every change of the CBO configuration should be carefully tested! Therefore, please, don't start tweaking your production system without testing the new settings on a test system. If it's really the only option you have, expect some problems…

Correctly Configuring the CBO

From time to time I have a heated discussion with people who say to me: "We don't need to spend time to individually configure the CBO for each database. We already have a set of initialization parameters that we use over and over again on all our databases." My first reply is, frequently, something like this: "Why do you think Oracle introduced almost two dozen initialization parameters that are CBO specific if a single set works well on all databases? They are not stupid, if such a 'magic configuration' exists, they will provide it by default and make those initialization parameters undocumented." Then I continue by carefully explaining that the 'magic configuration' doesn't exist because:

- Each application has its own requirements.
- Each system (hardware and software) has its own characteristics.

If the people in question are customers, usually, I remind them "You called me because you have performance problems, right? Actually in some situations the application isn't performing at its best, but also the database is responsible for the present situation... So, let's solve the problem."

That said, at least since Oracle9i, the CBO works well, i.e. it generates good execution plans for most[1] SQL statements. But be careful, this is only true, and I can never stress this enough, when the CBO is correctly configured and when the database has been designed to take advantage of all its features. Also notice that the configuration of the CBO includes not only the initialization parameters but the object and system statistics as well.

Core message #1: The CBO works well if it's correctly configured!

[1] Perfection is practically unrealizable in software development (and in almost any other activity you can think of as well…). This "law", even if neither we nor Oracle like it, applies to the CBO as well. Therefore you have to expect that a small percentage of the SQL statements will require a manual intervention, e.g. via hints, stored outlines or, in 10g only, SQL profiles.

Set the Right Parameter!

Each initialization parameter has been introduced for a specific reason, I'm sure that Oracle doesn't just randomly provide new initialization parameters! Understanding the impact of each initialization parameter on the CBO is essential for a successful configuration. Table 1 sums up the most important initialization parameter, grouped in 4 categories, related to the CBO.

Basic • OPTIMIZER_MODE • OPTIMIZER_FEATURES_ENABLE • OPTIMIZER_DYNAMIC_SAMPLING • OPTIMIZER_INDEX_COST_ADJ • OPTIMIZER_INDEX_CACHING • DB_FILE_MULTIBLOCK_READ_COUNT	Memory • WORKAREA_SIZE_POLICY • PGA_AGGREGATE_TARGET • HASH_AREA_SIZE • SORT_AREA_SIZE • BITMAP_MERGE_AREA_SIZE
Query transformation • QUERY_REWRITE_ENABLED • QUERY_REWRITE_INTEGRITY • STAR_TRANSFORMATION_ENABLED • OPTIMIZER_SECURE_VIEW_MERGING[2]	Miscellaneous • CURSOR_SHARING • SKIP_UNUSABLE_INDEXES • Parameters related to parallel processing • Many undocumented parameters…

Table 1: initialization parameters related to the CBO

When you understand how the CBO works and the role played by each of these initialization parameters, instead of tweaking the configuration randomly, you can:

1. Understand the current situation, e.g. why the optimizer has chosen an execution plan that is sub-optimal.
2. Define the goal to be achieved, i.e. which execution plan you want to achieve.
3. Find out which initialization parameters, or potentially which statistics, should be rectified to achieve that goal. In some situations it's also necessary to modify the SQL statement and/or the database design.

<u>Core message #2: Understanding how each initialization parameter changes the behavior of the CBO is essential.</u>

Configuration Roadmap

As I wrote in the previous section, I can't provide you with the "magic configuration". I can only show you how I proceed to do a configuration like this. Figure 1 sums up the main steps. Notice that only the initialization parameters that have to be set in almost all systems are shown in it. These are also the initialization parameters that I will further discuss in this paper. Of course, according to your needs, others have to be set as well (see Table 1 for a more exhaustive list).

[2] This parameter was added in Oracle Database 10g Release 2.

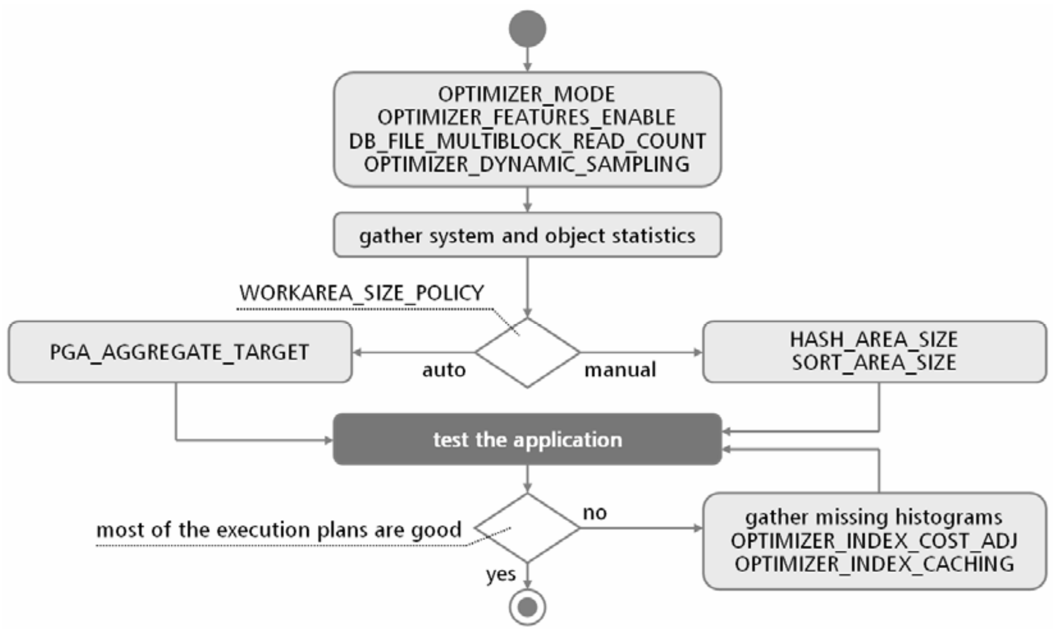
OPTIMIZER_MODE
OPTIMIZER_FEATURES_ENABLE
DB_FILE_MULTIBLOCK_READ_COUNT
OPTIMIZER_DYNAMIC_SAMPLING
gather system and object statistics
WORKAREA_SIZE_POLICY
PGA_AGGREGATE_TARGET
auto
manual
HASH_AREA_SIZE
SORT_AREA_SIZE
test the application
most of the execution plans are good
no
yes
gather missing histograms
OPTIMIZER_INDEX_COST_ADJ
OPTIMIZER_INDEX_CACHING

Figure 1: configuration roadmap main steps

First of all the basic initialization parameters (as I call them in Table 1) have to be set. Only exception is for the index related ones (i.e. OPTIMIZER_INDEX_COST_ADJ and OPTIMIZER_INDEX_CACHING). Notice that before gathering system statistics it's important to set DB_FILE_MULTIBLOCK_READ_COUNT. In fact this initialization parameter has a direct impact on the number of blocks read and, therefore, on the performance of multi-block read operations. The next step is to choose between manual and dynamic PGA management. Once this choice is made, you can test your application. During the test, for the components that don't provide the required performance, you have to collect the execution plans. By analyzing these execution plans you should be able to infer what the problem is. Notice that at this stage it's important to recognize general directions, not individual ones. Typical examples are that the CBO uses too many/few indexes or doesn't recognize the restrictions correctly. In the former case you could possibly fix the problem by changing the index related initialization parameters, in the former by gathering missing histograms.

According to Figure 1 the initialization parameters that are set before the test of the application cannot be changed afterwards. This isn't always the case. In fact if you cannot achieve good results by tweaking the index related initialization parameters it could be necessary to "restart" from the beginning. It's also worth mentioning that the index related initialization parameters have an indirect impact on system statistics as well and therefore, potentially, on all execution plans.

<u>Core message #3: Correctly configuring the CBO isn't an easy job.</u>
<u>This however is not a good reason for not doing it!</u>

Object Statistics

The CBO can generate good execution plans only if the object statistics, which describe the data stored in the database, are good as well, i.e. they must reflect existing data. Absent, out-of-date, or incorrect statistics can lead to poor execution plans[3]. The following object statistics are available.

Table statistics:

- Number of rows.
- Number of blocks below the high water mark.
- Average row length in bytes.

Column statistics:

- Number of distinct values.
- Number of NULL values.
- Data distribution (a.k.a. histograms).

Index statistics:

- Number of distinct keys.
- Height (the number of blocks that have to be read to get a leaf block).
- Number of leaf blocks.
- Clustering factor (this value indicates how many adjacent index entries don't refer to the same data block in the table).

For partitioned segments, object statistics exist at all levels. For example for a sub-partitioned table there are object statistics at the physical level (one set for each sub-partition) and at the logical levels (one set for each partition and one set for the table).

Up to Oracle9i Release 2, the DBA is responsible for the gathering of object statistics, i.e. by default no statistics are available. As of Oracle Database 10g a job named GATHER_STATS_JOB, which is based on the new scheduler, is automatically configured and scheduled during the creation of the database. This is a good thing, in fact in Oracle Database 10g the RBO no longer exists[4] and therefore statistics should be available for all objects. Of course the DBA is allowed to configure or even remove this job.

[3] In rare situations, to tune a SQL statement, object statistics have to be faked. This is mainly a workaround to a problem that cannot be solved another way. For this reason I will not discuss this topic in this paper.
[4] Actually the RBO is still available, but officially no longer supported.

It’s difficult to give general advice on the gathering of object statistics. Many factors influence the method used to keep them up-to-date. Here are some “rules” that I apply over and over again:

- Use the package DBMS_STATS[5].
- Gather statistics for tables, columns and indexes at the same time. This doesn’t mean you necessarily have to gather statistics with a single DBMS_STATS call, but only that it makes no sense, for example, to gather table statistics today, column statistics in 10 days and index statistics in a month.
- To speed up the gathering of statistics, use small estimate percentages: values less than 10% are usually good. For large tables even 0.5%, 0.1% or less could be fine. An interesting feature of DBMS_STATS is that the specified value is increased if necessary (that is it’s a minimum value). Therefore, if the gathering of statistics is performed at database or schema level, the estimate percentage should be chosen for the bigger table. Listing 1 shows an example of this behavior. If the parameter CASCADE is set to TRUE, it’s also possible to see different estimate percentages for the same table, e.g. 10% for table and columns statistics, and 100% for the indexes.
- It makes no sense to gather statistics on static data over and over again. Therefore table monitoring should be activated and the gathering of statistics should be performed only when a relevant part of the data is changed (by default a threshold of 10% is used when the parameter OPTIONS is set to GATHER STALE). In Oracle9i this is carried out with DBMS_STATS.ALTER_SCHEMA_TAB_MONITORING. In Oracle Database 10g by setting the initialization parameter STATISTICS_LEVEL to TYPICAL (this is the default). To use a different threshold or to apply the monitoring information for a statistics gathering at table level, the data provided in ALL_TAB_MODIFICATIONS can be directly accessed.
- For partitioned tables gather statistics at all levels.
- For large tables gather statistics in parallel.
- Histograms are essential for all columns referenced in WHERE clauses that contain skewed data. Notice that they are useful on non-indexed columns as well! For simplicity use SIZE SKEWONLY. If it takes too much time try SIZE AUTO[6]. If it’s still too slow or the chosen number of buckets is not good (or the needed histogram isn’t created at all), manually specify the list of columns.

```
SQL> exec dbms_stats.delete_schema_stats(user)

SQL> BEGIN
  2   dbms_stats.gather_schema_stats(
  3     ownname=>user,
  4     cascade=>TRUE,
  5     estimate_percent=>1,
  6     method_opt=>'for all columns size 1');
  7  END;
  8  /
```

[5] As of Oracle Database 10g the gathering of object statistics through the statement ANALYZE is supported for backward compatibility only. Therefore I will not discuss ANALYZE in this paper.

[6] SIZE AUTO gathers less histograms than SIZE SKEWONLY because it checks through SYS.COL_USAGE$ if a column has already be used in the WHERE clause. This is good only if the application has already been used on that specific database, otherwise no histograms at all are created.

```
SQL> SELECT table_name, num_rows, round(sample_size/num_rows*100) "%"
  2  FROM user_tables
  3  WHERE sample_size > 0
  4  ORDER BY table_name;

TABLE_NAME                       NUM_ROWS          %
------------------------------ ---------- ----------
CAL_MONTH_SALES_MV                     48        100
CHANNELS                                5        100
COSTS                               82260         10
COUNTRIES                              23        100
CUSTOMERS                           55590         10
FWEEK_PSCAT_SALES_MV                11105         40
PRODUCTS                               72        100
PROMOTIONS                            503        100
SALES                              920300          1
SUPPLEMENTARY_DEMOGRAPHICS           4500        100
TIMES                                1826        100
```

Listing 1: DBMS_STATS increases the estimate percentage if necessary (the gathering has been performed with 1% but this estimate percentage has been used for a single table, the biggest)

Now, based on these recommendations, if you have a database without particular requirements, a command like the one shown in Listing 2 is a good starting point. Since GATHER STALE is specified, monitoring has to be activated on the target schema.

```
dbms_stats.gather_schema_stats(
    ownname          => user,
    estimate_percent => 5,
    cascade          => TRUE,
    method_opt       => 'FOR ALL COLUMNS SIZE SKEWONLY',
    options          => 'GATHER STALE'
);
```

Listing 2: a simple command to gather statistics at schema level

Even if such a command works fine in many situations, on more challenging systems we are often forced to have a parameterization at object level (at least for the most important tables). This usually means that we have to store the DBMS_STATS parameters of each object in a help table and then, through some code we have to write as well, to gather the statistics individually for each object, i.e. not at schema or database level.

Every strategy has pro and cons. How should it be chosen? In my opinion it makes sense to start with the simplest one. Then, if the requirements are not fulfilled, a more complex strategy should be taken into account. Simple problems usually require simple solutions!

As of Oracle Database 10g all procedures that gather optimizer statistics no longer have hard coded default values. Instead, they are stored in the data dictionary. The idea is to set the default values according to your needs. Then, when DBMS_STATS is called, there is no need to specify all parameters. Listing 3 shows the default values in 10.2. To get and set the default values the DBMS_STATS procedures GET_PARAM and SET_PARAM have to be used.

```
SQL> SELECT sname, nvl(to_char(sval1),spare4) value
  2  FROM sys.optstat_hist_control$;

SNAME                VALUE
-------------------- ------------------------------
SKIP_TIME
STATS_RETENTION      31
TRACE                0
DEBUG                0
SYS_FLAGS            1
CASCADE              DBMS_STATS.AUTO_CASCADE
ESTIMATE_PERCENT     DBMS_STATS.AUTO_SAMPLE_SIZE
DEGREE               NULL
METHOD_OPT           FOR ALL COLUMNS SIZE AUTO
NO_INVALIDATE        DBMS_STATS.AUTO_INVALIDATE
GRANULARITY          AUTO
AUTOSTATS_TARGET     AUTO
```

Listing 3: DBMS_STATS default values stored in the data dictionary

Another interesting new feature of Oracle Database 10g is the ability to lock object statistics. This is especially useful for freezing statistics on selected objects when the gathering is performed at schema or database level. To (un)lock object statistics, the DBMS_STATS procedures with the prefix (UN)LOCK are available.

System Statistics

The CBO used to base its estimations on the number of I/O needed to execute a SQL statement (called “I/O cost model”). The main deficiency of this method is that single-block reads and multi-block reads are equally costly[7]. Therefore, full scans are artificially favored. Up to Oracle8i, especially in OLTP systems, we used to change OPTIMIZER_INDEX_COST_ADJ and OPTIMIZER_INDEX_CACHING. In fact the default values used to be appropriate for reporting systems and data warehouses only. As of Oracle9i, to address this flaw, a new method, called “CPU cost model”, has been implemented. To enable the CPU cost model additional information about the performance of the system where Oracle runs, called system statistics, have to be provided to the CBO. Essentially, system statistics supply the following information:

- Performance of the I/O subsystem.
- Average I/O size for multi-block read operations.
- Performance of the CPU.

There are two kinds of system statistics, “without workload” and “with workload”. Listing 4 shows how to get them from the data dictionary.

[7] Common sense suggests that reading a single block should be faster than reading multiple blocks. Strangely enough, this is not always true in reality! Anyway, the important thing is to recognize that there is a difference.

Without workload (only available in Oracle Database 10g):

- CPUSPEEDNW: CPU speed in millions of operations per second.
- IOSEEKTIM: I/O seek time [ms].
- IOTFRSPEED: I/O transfer speed [bytes/s].

With workload:

- CPUSPEED: CPU speed in millions of operations per second.
- SREADTIM: single-block read time [ms].
- MREADTIM: multi-block read time [ms].
- MBRC: average number of blocks read during a multi-block read operation.
- MAXTHR: maximum I/O throughput [bytes/s].
- SLAVETHR: maximum slave throughput [bytes/s].

```
SQL> SELECT pname, pval1 FROM sys.aux_stats$ WHERE sname = 'SYSSTATS_MAIN';

PNAME                  PVAL1
--------------- ------------
CPUSPEEDNW           886.233
IOSEEKTIM                 10
IOTFRSPEED              4096
CPUSPEED                 928
SREADTIM               8.138
MREADTIM              16.799
MBRC                       9
MAXTHR             162238464
SLAVETHR              277504
```

Listing 4: system statistics stored in the data dictionary

System statistics without workload aren't really helpful. The reason is simple, they provide little (in Oracle9i no) information about the system… Nevertheless, for the sake of completeness, in Listing 5 the command used to gather them is shown. As of Oracle Database 10g they are automatically gathered at startup if they don't exist.

```
dbms_stats.gather_system_stats('noworkload');
```

Listing 5: gathering system statistics without workload

System statistics with workload is what we are looking for. They should be gathered when the system is charged with a common workload, e.g. you could run your test suites. During the gathering the system isn't charged at all (at least for the gathering itself…). In fact DBMS_STATS just stores some information at the beginning and at the end of the gathering. Then, with these values, it computes the system statistics. For this reason the gathering period should be quite long, i.e. not just a few minutes, in this way they should be much more representative. They are gathered with the command shown in Listing 6.

```
dbms_stats.gather_system_stats(
  gathering_mode => 'interval',
  interval       => 30
);
```

Listing 6: gathering system statistics with workload over the next 30 minutes

When system statistics are available the CBO computes two costs: I/O and CPU. Then, with the following formula[8], it sums them. Listing 7 gives and an example for such a computation.

$$Cost \approx I/O\,Cost + \frac{CPU\,Cost}{CPUSPEED \cdot SREADTIM \cdot 1000}$$

```
SQL> EXPLAIN PLAN FOR SELECT * FROM sh.sales;

SQL> SELECT cost, cpu_cost, io_cost FROM plan_table WHERE id = 0;

      COST   CPU_COST    IO_COST
---------- ---------- ----------
       706  261078827        507

SQL> SELECT pname, pval1
  2  FROM sys.aux_stats$
  3  WHERE pname IN ('CPUSPEED', 'SREADTIM');

PNAME                                PVAL1
------------------------------ ----------
SREADTIM                             2.209
CPUSPEED                               593

SQL> SELECT 507+261078827/(593*2.209*1000) cost FROM dual;

      COST
----------
706.306399
```

Listing 7: computation of the cost based on I/O cost, CPU cost and system statistics

Since system statistics makes the CBO aware of the system where Oracle is running, they are essential for a successful configuration. My advice, therefore, is to use them[9]. To have some stability I usually recommend freezing them. Of course, in case of major hardware or software changes, they should be recomputed and, therefore, the whole configuration should be checked.

[8] This formula has been taken from “Oracle9i Database Performance Tuning Guide and Reference”. Notice that the formula has disappeared from the manuals in 10g.

[9] As for any new features, in some situations we are not able to use them because of bugs.

One last bit of information, as of Oracle Database 10g the hint NO_CPU_COSTING can be specified to use the I/O cost model when system statistics have been gathered. The unsupported way to do it, even in 9i, is to set the undocumented initialization parameter _OPTIMIZER_COST_MODEL to IO.

OPTIMIZER_MODE (OM)

This is the most important initialization parameter. Unfortunately, too often, it's not set, i.e. its default value is used.
Up to Oracle9i Release 2 the default value is CHOOSE. This means that if object statistics are available for at least one of the objects referenced in the SQL statement that has to be tuned, ALL_ROWS is used; otherwise RULE. In any case, in some situations, even when no object statistics are available, Oracle "forces" ALL_ROWS. This is because the rule-based optimizer (RBO) doesn't support objects added after Oracle7 (e.g. for partitioned tables).
As of Oracle Database 10g Release 1 the RBO has been deprecated and therefore the new default value is ALL_ROWS.

To choose the value of OM you have to ask yourself if it's more important that the CBO tunes SQL statements for fast delivery of the first or the last row.

- If fast delivery of the <u>last</u> row is important, ALL_ROWS should be used (typical in reporting systems and data warehouses).
- If fast delivery of the <u>first</u> row(s) is important, FIRST_ROWS_n (where "n" is [1 | 10 | 100 | 1000] rows) should be used (typical for OLTP systems).

OPTIMIZER_FEATURES_ENABLE (OFE)

In each database version Oracle introduces or enables new CBO features. OFE can be used to set which "version" of the CBO should be used. Valid values are database versions like 8.1.7, 9.2.0 or 10.1.0. The default is the current database version. Notice, however, that not all new features are enabled/disabled by this initialization parameter[10]. This means that even if you set it to 8.1.7, for example in Oracle Database 10g, you won't get the 8.1.7 optimizer! This is mainly due to some transformations that are applied to the SQL statements before sending them to the CBO or RBO.

It could be useful to set OPTIMIZER_FEATURES_ENABLE when an application is upgraded to a new database version, otherwise leave it at the default value.

[10] A complete list of the features which are enabled/disabled by each setting is available in the "Oracle Database Reference" manual.

DB_FILE_MULTIBLOCK_READ_COUNT (DFMBRC)

DFMBRC specifies the maximum number of blocks that Oracle reads during a multi-block operation (e.g. full table scan). In fact there are three common situations that lead to multi-block reads that are smaller than the value specified by DFMBRC:

- Oracle reads segment headers with single-block reads.
- Oracle never does an I/O that spans more extents.
- Oracle never reads, except for direct I/O, a block that is already in the buffer cache.

Multi-block reads are a performance feature. Therefore, DFMBRC should be set to achieve the best performance. To do so is important to recognize that higher values don't provide better performance in all cases and, in addition, that it makes no sense to exceed the maximum physical I/O size. A simple full table scan with different values gives useful information about the impact of this initialization parameter and, therefore, assists in finding the "best" value. As you can see in Figure 2 different systems have different characteristics.

- System 1: The performance increases with a larger I/O size. The gain is important for values up to 8-10. Larger values give small benefits.
- System 2: Values up to 16 perform poorly (gain < 10%). By switching from 16 to 17, the gain increases by 30%! Values larger than 17 provide little benefit. For this system values smaller than 17 should be avoided.
- System 3: The gain is important for values up to 8. For values between 8 and 16, the gain is stable. By switching from 16 to 17 the gain decreases by 16%! For this system values greater than 16 should be avoided.
- System 4: Strangely enough, the performance of this system is independent of the I/O size. Notice that this is also the system with the best throughput.

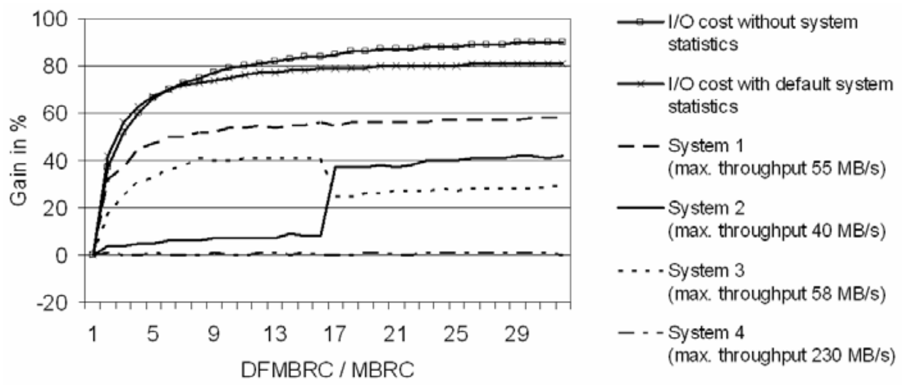

Figure 2: impact of the I/O size on CBO estimations and performance for a full table scan

When system statistics with workload are not used, DFMBRC has a direct impact on the cost of multi-block operations. Therefore, too high values lead to excessive full scans.

- Without system statistics the costing formula[11] is the following (see Figure 2 for a graphical representation):

$$I/O\,Cost\,FS \approx \frac{Blocks}{1.6765 \cdot DFMBRC^{0.6581}} + 1$$

- With system statistics the costing formula is the following (see Figure 2 for a graphical representation):

$$I/O\,Cost\,FS \approx \frac{Blocks}{MBRC} \cdot \frac{MREADTIM}{SREADTIM} + 1$$

When system statistics without workload are used the value for MBRC, MREADTIM and SREADTIM are not available. Therefore they are derived with the following formula.

$$MBRC = DFMBRC$$

$$MREADTIM = IOSEEKTIM + \frac{DFMBRC \cdot BlockSize}{IOTFRSPEED}$$

$$SREADTIM = IOSEEKTIM + \frac{BlockSize}{IOTFRSPEED}$$

As of Oracle Database 10g Release 2 it's possible to set Oracle to automatically tune the value of DFMBRC. To use this function simply don't set DFMBRC. Notice that if I/O costing is used the CBO, instead of using the automatically tuned value of DFMBRC, uses 8 to estimate the cost of multi-block read operations.

OPTIMIZER_DYNAMIC_SAMPLING (ODS)

The CBO used to base its estimations on object statistics found in the data dictionary only. With dynamic sampling some statistics can be gathered during the parse phase as well. This means that some queries are executed against the referenced objects to gather additional information. The statistics gathered by dynamic sampling aren't stored in the data dictionary, they are simply shared at cursor level.

The value of the initialization parameter ODS specifies how and when dynamic sampling is used. Table 2 sums up the available levels. The default values are the following:

- If OPTIMIZER_FEATURES_ENABLE ≥ 10.0.0 → 2
- If OPTIMIZER_FEATURES_ENABLE = 9.2.0 → 1
- If OPTIMIZER_FEATURES_ENABLE ≤ 9.0.1 → 0

Level 1 and 2 aren't very useful. In fact, tables and indexes should have up-to-date statistics! An exception is when temporary tables are used. In fact, usually, no statistics are available for them. Level 3 and up are useful for improving selectivity estimations of "complex"

[11] This formula was originally proposed by Wolfgang Breitling in his well-known paper "A Look Under the Hood of CBO: The 10053 Event".

predicates. If the CBO isn't able to make correct estimations, start by setting it to 4, otherwise set it to 2.

Level	On which table?	# Blocks[12]	
		Param	Hint
0	Dynamic sampling disabled	0	0
1	All non-analyzed tables, if at least one table • Is part of a join, subquery or non-mergeable view • Has no index • Has more blocks than the number of blocks used for the sampling	32	32
2	All non-analyzed tables	64	64
3	All tables which fulfill the level-2 criterion plus all tables for which a standard selectivity estimation is used	64	128
4	All tables which fulfill level-3 criteria plus all tables which have single table predicates referencing more than two attributes	64	256
5		64	512
6		128	1024
7		256	2048
8		1024	4096
9		4096	8192
10		All	All

Table 2: dynamic sampling levels

PGA Management

It's possible to choose between two methods to manage the PGA:
- Manual: the DBA has full control over the size of the PGA.
- Automatic: the DBA delegates the management of the PGA to Oracle.

Except in the following situations it's recommended to use automatic PGA management:
- Very large PGAs are needed[13] (>100MB, see Metalink note 147806.1).
- Fine tuning is needed.
- Shared server (former MTS) is used (9i only).

It is also possible to enable automatic PGA management at system level and then, for special requirements, to switch to manual PGA management at session level.

Usually a larger PGA makes merge/hash joins and sort operations faster. Therefore, you should devote the "unused" memory that is available on the system to it. But, be careful when you change it. The PGA size in fact has an influence on the costs estimated by the CBO as well.

[12] Strangely enough when dynamic sampling is enabled at session or statement level, the values used for the maximum number of blocks are different. No idea why!
[13] The undocumented parameter _PGA_MAX_SIZE may help in some situations. However it's undocumented and therefore not supported, i.e. you are on your own if you want to use it!

To enable manual PGA management the initialization parameter WORKAREA_SIZE_POLICY must be set to MANUAL. Then the initialization parameters HASH_AREA_SIZE and SORT_AREA_SIZE should be set as well. It's practically impossible to give advice about their value. Nevertheless, here some general rules:

- Usually a minimal value of 512KB to 1MB should be used.
- For small PGAs, to take advantage of hash join, the HASH_AREA_SIZE should be at least 3-4 times the SORT_AREA_SIZE.

HASH_AREA_SIZE and SORT_AREA_SIZE specifies the maximum amount of memory that can be used by each process.

To enable auto PGA management the initialization parameter WORKAREA_SIZE_POLICY must be set to AUTO. Contrary to manual PGA management, the initialization parameter PGA_AGGREGATE_TARGET specifies the total amount of memory use for all PGAs, i.e. not for a single process. In some situations the PGA_AGGREGATE_TARGET could also be exceeded. In fact it's not a maximum, but a target! This usually happens when a too small value is specified.

OPTIMIZER_INDEX_COST_ADJ (OICA)

This initialization parameter is used to change the cost of table accesses through index scans. Valid values go from 1 to 10000, the default is 100. Values greater than 100 make index scans more expensive and, therefore, favor full scans. Values lower than 100 make index scans less expensive.

The correction applied to the index range scan costing is shown in the following formula:

$$I/O\,Cost \approx \left(BLevel + \left(LeafBlocks + ClusteringFactor\right) \cdot Selectivity\right) \cdot \frac{OICA}{100}$$

The correction applied to the index unique scan costing is shown in the following formula:

$$I/O\,Cost \approx \left(BLevel + 1\right) \cdot \frac{OICA}{100}$$

This initialization parameter flattens costs and makes the clustering factor much less significant. Listing 8 shows exactly that problem, let's describe what's going on:

- The first query shows index statistics. The only difference, between the two indexes, is the clustering factor. Actually I2 has a much better clustering factor than I1.
- The second query shows the definition of both indexes.
- Set OPTIMIZER_INDEX_COST_ADJ to 100.
- Because of the better clustering factor, as expected the test query uses I2 to retrieve data.
- Set OPTIMIZER_INDEX_COST_ADJ to 10.
- The test query uses I1 to retrieve data, even if I2 has a much better (lower) clustering factor!
- Rename the index I1 to I3.
- The test query uses I2 to retrieve data!!! In other words the execution plans are dependent on the name of the indexes. This happens because the CBO, when it is not able to decide which index should be used based on costs, takes the first one in alphabetical order.

Therefore, small values should be carefully specified! With system statistics the default value is usually good.

```
SQL> SELECT index_name, num_rows, distinct_keys,
  2         blevel, leaf_blocks, clustering_factor
  3  FROM user_indexes
  4  WHERE table_name = 'T';

INDEX_NAME   NUM_ROWS DISTINCT_KEYS     BLEVEL LEAF_BLOCKS CLUSTERING_FACTOR
---------- ---------- ------------- ---------- ----------- -----------------
I1              10000          1000          1          21              5222
I2              10000          1000          1          21               437

SQL> SELECT index_name, column_name
  2  FROM user_ind_columns
  3  WHERE index_name IN ('I1','I2');

INDEX_NAME           COLUMN_NAME
-------------------- --------------------
I1                   COL1
I2                   COL2

SQL> SET AUTOTRACE TRACEONLY EXPLAIN

SQL> ALTER SESSION SET OPTIMIZER_INDEX_COST_ADJ=100;

SQL> SELECT * FROM t WHERE col1 = 11 AND col2 = 11;

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=1 Bytes=62)
  TABLE ACCESS (BY INDEX ROWID) OF 'T' (TABLE) (Cost=2 Card=1 Bytes=62)
    INDEX (RANGE SCAN) OF 'I2' (INDEX) (Cost=1 Card=10)

SQL> ALTER SESSION SET OPTIMIZER_INDEX_COST_ADJ=10;

SQL> SELECT * FROM t WHERE col1 = 11 AND col2 = 11;

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1 Card=1 Bytes=62)
  TABLE ACCESS (BY INDEX ROWID) OF 'T' (TABLE) (Cost=1 Card=1 Bytes=62)
    INDEX (RANGE SCAN) OF 'I1' (INDEX) (Cost=1 Card=10)

SQL> ALTER INDEX i1 RENAME TO i3;

SQL> SELECT * FROM t WHERE col1 = 11 AND col2 = 11;

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1 Card=1 Bytes=62)
  TABLE ACCESS (BY INDEX ROWID) OF 'T' (TABLE) (Cost=1 Card=1 Bytes=62)
    INDEX (RANGE SCAN) OF 'I2' (INDEX) (Cost=1 Card=10)
```

Listing 8: example of the instability introduced by setting a low OICA

OPTIMIZER_INDEX_CACHING (OIC)

This initialization parameter is used to specify the expected amount (in percent) of index blocks cached in the buffer cache during nested loop joins and indexed inlist iterators. It is important to note that the value of this initialization parameter is used by the CBO to adjust its estimations, i.e. it doesn't specify how much of the indexes are actually cached by Oracle! Valid values go from 0 to 100, the default is 0. Values greater than 0 reduce the cost of nested loop joins and indexed inlist iterators.

The correction applied to the index range scan costing is shown in the following formula:

$$I/O\,Cost \approx \left(BLevel + LeafBlocks \cdot Selectivity\right) \cdot \left(1 - \frac{OIC}{100}\right) + ClusteringFactor \cdot Selectivity$$

The correction applied to the index unique scan costing is shown in the following formula:

$$I/O\,Cost \approx BLevel \cdot \left(1 - \frac{OIC}{100}\right) + 1$$

If you set OICA and OIC at the same time, just multiply the results of the last two formulas by OICA/100.
As for OICA, with system statistics the default value is usually good.

Conclusion

Oracle offers plenty of initialization parameters for the configuration of the CBO. In addition there are various methods for collecting system and object statistics. To take full advantage of this powerful piece of software, you have to understand the impact of initialization parameters and statistics on the CBO and invest enough time to perform an accurate configuration.

Feel free to contact me if you have any comments or questions about this paper. And, of course, I'm completely at your disposal to make such a configuration on one of your systems. It's my job!

Address

Christian Antognini
Trivadis AG
Europa-Strasse 5
CH-8152 Glattbrugg / Zürich

Telephone	+41 44 808 70 20
Fax	+41 44 808 70 21
E-Mail	christian.antognini@trivadis.com
Internet	www.trivadis.com