The Effects of optimizer_index_cost_adj and optimizer_index_caching on Access Plans

Wolfgang Breitling
breitliw@centrexcc.com

PREREQUISITE:

Presentation "Introduction to the Cost Based Optimizer Trace" or equivalent knowledge. A desire to better understand the Cost Based Optimizer.

OBJECTIVES:

ABSTRACT:
Two init.ora parameters are widely touted as the “silver bullet ” to end all CBO bad access path choices: OPTIMIZER_INDEX_COST_ADJ (OICA) AND OPTIMIZER_INDEX_CACHING (OIC). Everyone seems to have their own favorite numbers for them . Tim Gorman explains why and how to derive the numbers from formulas using Oracle system-wide statistics. This presentation shows where and how the setting of these parameters affect the index access cost and access path composition, contrasting OICA with the SREADTIM and MREADTIM values of the Oracle 9 and 10 system statistics. The goal is to come to a better understanding of the CBO’s index access cost “formulas”.

Who am I

- Independent consultant since 1996
  - specializing in Oracle and Peoplesoft setup, administration, and performance tuning
- Member of the Oaktable Network
- 25+ years in database management
  - DL/1, IMS, ADABAS, SQL/DS, DB2, Oracle
- OCP certified DBA - 7, 8, 8*i*, 9*i*
- Oracle since 1993 (7.0.12)
- Mathematics major at University of Stuttgart

Agenda

- Show how different setting of o_i_c_a and o_i_c change the CBO's cost calculation
- Show one case where the CBO refuses to be lied to
- Show how the changed costs translate into access path transformations
- Contrast the o_i_c_a parameters to the effect of system statistics

All the observations stem from Oracle on Windows and Linux Redhat ES3. In the past I have not seen any differences in the way the CBO "behaves" between different platforms. Actually I often do explain plan analysis for unix based databases on an NT based "sandbox".

Index Access Costs

Unique scan	blevel + 1
Fast full scan	leaf_blocks / k
Index-only	blevel + FF_i * leaf_blocks
Range scan	blevel + FF_i * leaf_blocks + FF_t * clustering_factor

These are the four basic index access methods and their cost breakdown. There are index access subclasses. A full taxonomy of index accesses would be beyond the scope of this presentation. This is a partial list, which may be version dependent, based on what I have observed in 10053 traces:

index (equal)
index (eq-unique)
index (iff)
index (index-only)
index (join index)
index (join stp)
index (min/max)
index (no sta/stp keys)
index (scan)
index (stp-guess)
index (unique)

Single Table Access Path and o_i_c_a

```
OPTIMIZER_INDEX_COST_ADJ = 100
  Access path: tsc                                Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891
                                       BEST_CST:           4698.00  PATH: 2

OPTIMIZER_INDEX_COST_ADJ = 10
  Access path: tsc                                Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891
                                       BEST_CST:            689.10  PATH: 4

OPTIMIZER_INDEX_COST_ADJ = 1
  Access path: tsc                                Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891
                                       BEST_CST:             68.91  PATH: 4

OPTIMIZER_INDEX_COST_ADJ = 1000
  Access path: tsc                                Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891
                                       BEST_CST:           4698.00  PATH: 2
```

Setting event 10183 prevents roundi

That's what is done in experiments: change a parameter and observe the effect of the change on the system.

By setting oica=10, 1/10th the original, best_cst becomes 689.10, exactly 1/10th of the index (equal) access cost of 6891 and the path changes from 2 (tablescan) to 4 (index access).

On the other side of the scale, setting oica = 1000, 10 times the original (or any value > 100) does apparently have no effect at all. We can extrapolate that it actually does increase any index access costs by the corresponding factor. The 10053 trace does not always show intermediate calculations. In this, as well as other, instances only the final result (best_cst) is shown, which didn't change.

Single Table Access Path and o_i_c_a

Index access costs get "discounted" by applying a factor of

optimizer_index_cost_adj/100

to the calculated index costs before deciding the lowest-cost single table access path.

That should not come as a surprise. That cat has been out of the bag for some time, even the name suggests it.

Single Table Access Path and o_i_c_a

Since 4698/6891 = 0.681759, it is to be expected that the adjusted "index (equal)" access cost, and with it the single table access path to fall below the tsc access cost – which remains unadjusted – between o_i_c_a = 69 and o_i_c_a = 68:

```
OPTIMIZER_INDEX_COST_ADJ = 100
  Access path: tsc                                 Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891
                                       BEST_CST:            4698.00  PATH: 2

OPTIMIZER_INDEX_COST_ADJ = 69
  Access path: tsc                                 Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954     69% of 6891
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891  [4754.79]
                                       BEST_CST:            4698.00  PATH: 2

OPTIMIZER_INDEX_COST_ADJ = 68
  Access path: tsc                                 Resc:    4698
  Access path: index (no sta/stp keys) RSC_CPU: 0  RSC_IO: 27954
  Access path: index (equal)           RSC_CPU: 0  RSC_IO:  6891
                                       BEST_CST:            4685.88  PATH: 4
```

The next step in formulating a theory is to make predictions based on the theory and test if the prediction holds. If it does, the theory is tenable.

Joins and o_i_c_a

o_i_c_a affects joins in 2, perhaps 3 ways

- adjusted cost of the outer table from carried single table access best_cst
- adjusted NL index access costs
- adjusted index access cost instead of a sort in SM join

The cost reduction for single table access paths is only the beginning. The next step is to look at the effects of oica on join costs.

The first cost effect is what we just saw for single table access path costs. We'll cover pont two next. Point 3 seems logical. However, since I have not seen any cases in the tests I have done I have to leave it as hypothetical.

o_i_c_a and NL Joins

```
OPTIMIZER_INDEX_COST_ADJ = 100
Outer table: cost: 4698  cdn: 144  rcz: 43
Inner table:
  Access path: tsc                         Resc:  2541  Join: 370602
  Access path: index (unique)              RSC_IO:   2  Join:   4986
  Access path: index (scan)                RSC_IO:   3  Join:   5130
  Access path: index (no sta/stp keys)     RSC_IO:3153  Join: 458730
  Access path: index (scan)                RSC_IO: 271  Join:  43722
  Access path: index (eq  unique)          RSC_IO:   2  Join:   4986
Best NL cost: 4986
                     No fractional costs shown despite event 10183
OPTIMIZER_INDEX_COST_ADJ = 10
Outer table: cost: 689  cdn: 144  rcz: 43          689 + 144*2 * 10/100
  Inner table:
  Access path: tsc                         Resc:  2541  Join: 366593
  Access path: index (unique)              RSC_IO:   2  Join:    718   14.4%
  Access path: index (scan)                RSC_IO:   3  Join:    732   14.3%
  Access path: index (no sta/stp keys)     RSC_IO:3153  Join:  46092   10.0%
  Access path: index (scan)                RSC_IO: 271  Join:   4592   10.5%
  Access path: index (eq  unique)          RSC_IO:   2  Join:    718   14.4%
Best NL cost: 718
```

Continuing with the same table we used for single table access paths, we examine the different ways to join it to the next table in an NL join.

Note that we no longer get fractional costs despite event 10183 being set.

We can see that the oica factor is also applied to the index access cost of the joined table.

In the NL join, the “discount factor” is magnified by the estimated cardinality of the outer table, i.e., oica may have marginal effect on the single table access path cost, but its impact in a NL join is multiplied by the outer table cardinality. The higher the outer table cardinality, the higher the cost reduction effect.

o_i_c_a and HA and SM Joins

Beyond possibly reduced access costs of outer and inner tables, there is no further effect of optimizer_index_cost_adj on HA joins

I have not observed any cases where an index on the inner table in an SM join was used to avoid the sort and was discounted, but it is certainly possible.

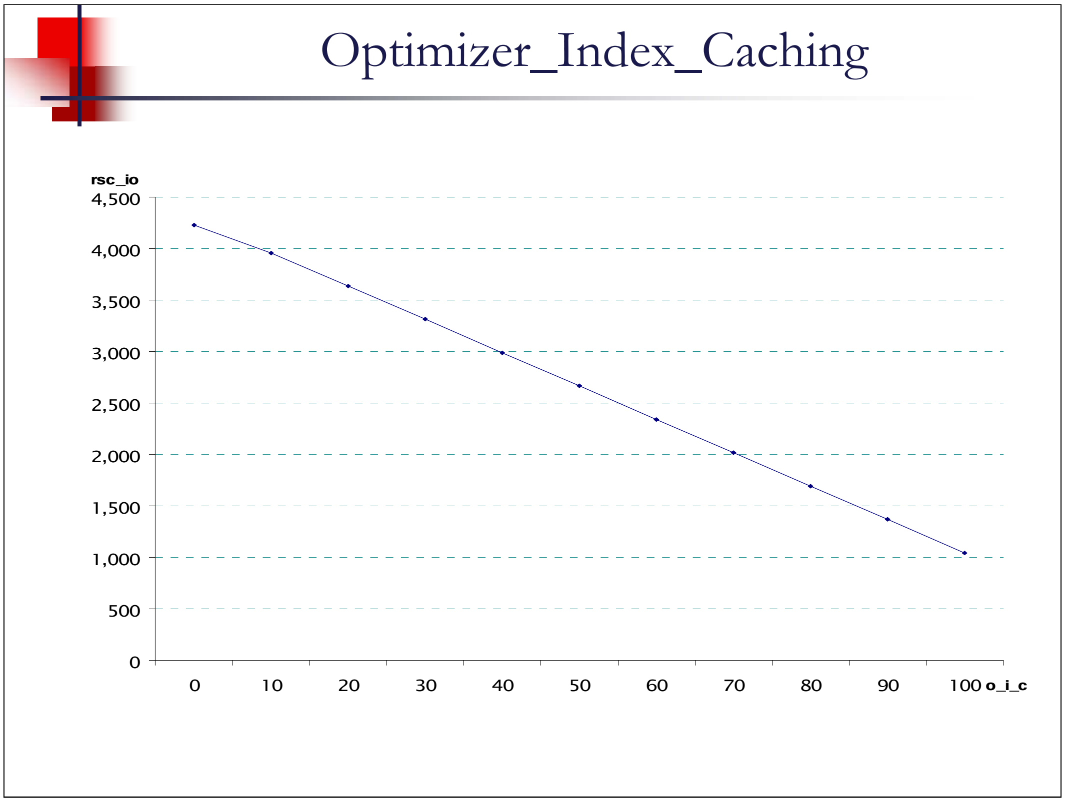
Optimizer_Index_Caching

```
LVLS: 2   #LB: 3178  CLUF: 21958

Outer table: cost: 7387  cdn: 429  rcz: 23
   0 Access path: index (no sta/stp keys) RSC_IO: 4226 Join: 1820341
  10 Access path: index (no sta/stp keys) RSC_IO: 3959 Join: 1705798
  20 Access path: index (no sta/stp keys) RSC_IO: 3635 Join: 1566802
  30 Access path: index (no sta/stp keys) RSC_IO: 3312 Join: 1428235
  40 Access path: index (no sta/stp keys) RSC_IO: 2988 Join: 1289239
  50 Access path: index (no sta/stp keys) RSC_IO: 2664 Join: 1150243
  60 Access path: index (no sta/stp keys) RSC_IO: 2341 Join: 1011676
  70 Access path: index (no sta/stp keys) RSC_IO: 2017 Join:  872680
  80 Access path: index (no sta/stp keys) RSC_IO: 1694 Join:  734113
  90 Access path: index (no sta/stp keys) RSC_IO: 1370 Join:  595117
 100 Access path: index (no sta/stp keys) RSC_IO: 1046 Join:  456121
```

On to OIC. Again we vary it and observe what changes as a result.

- First observation, not shown here: Single Table Access Costs are not affected, only index access costs within NL joins.
  In effect, the CBO does not recognize index caching benefits between sql statements, only within a statement.
- There both, rsc_io (the index access cost) and the total join cost, clearly show a decrease.
  Of course the two are correlated.



When charting the rsc_io values against the oic parameter it is clear that the relationship is largely linear.

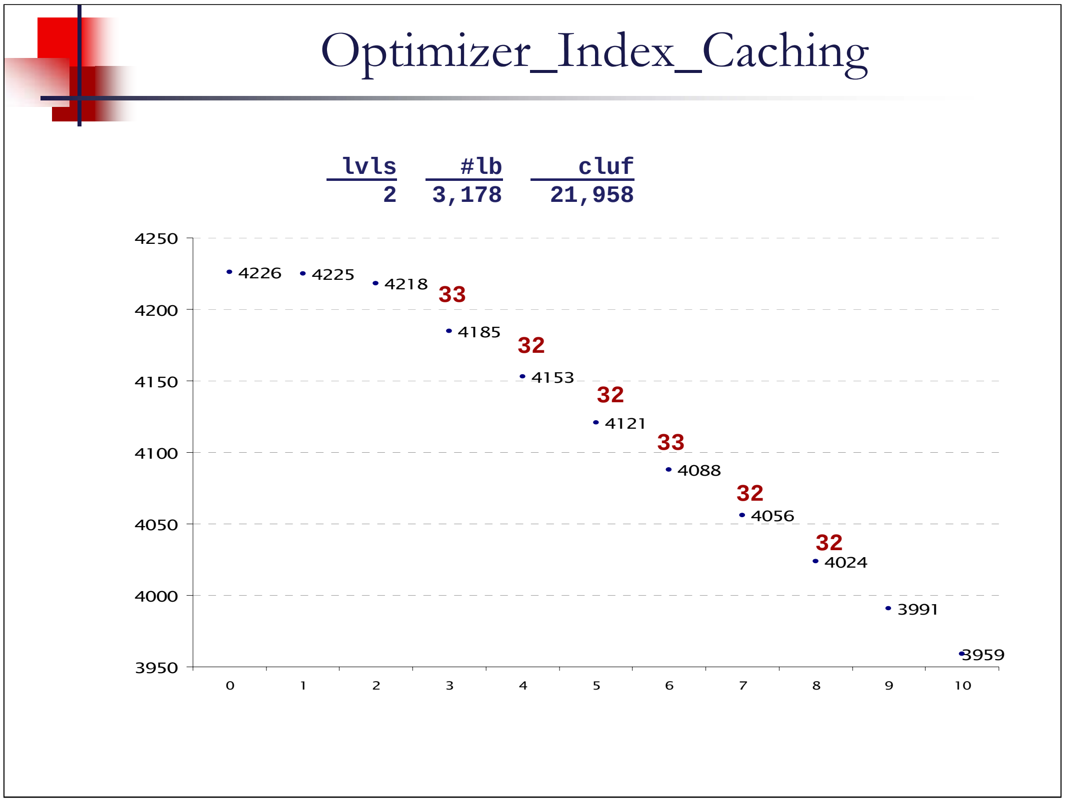
If one looks closely one can see that the first section has a slightly lower slope than the rest. Of course the chart was obtained from test where oic was varied in intervals of 10 points. We don't know for sure what happends to the costs between those intervals. We'll look at the first interval, 0..10, in detail in a moment.

Optimizer_Index_Caching

$$rsc\_io \approx lvls + (100 - oic)/100 * ix\_sel * \#lb + tb\_sel * cluf$$

$$join = \$_{outer} + cdn_{outer} * rsc\_io_{inner}$$

From the prior chart we can derive this adjusted formula for the index access cost. Here also, the cost reduction by the oic factor in the rsc_io cost formula is magnified by the outer table cardinality in the NL join cost calculation.



As promised, here is a closer look a oic 0..10:

- From oic=0 to oic=1 the cost drop by 1. This is actually a coincidence of the fact that the index height is 2. The costs drop by height-1 from oic=0 to oic=1.
- In this example of a smallish index the actual decline starts at oic=2. With larger indexes there are longer intervals where the cost stays at the initial cost – (height-1). The longest I have observed in my tests is up to oic=4 with the declines beginning at 5.
- Once the decline starts it is slightly steeper than the formula on the prior page suggests. Given the formula we ought to expect a slope of -31.78 (#lb/100) but the slope is between -32 and -33, approximately at -32.3. We'll se another, more significant, deviation of the cost behavior with increasing oic values in a moment.

3178 / 98 = 32.42 !!

Index Access Costs

Unique scan	blevel + 1
	1 or even 0
Fast full scan	leaf_blocks / k
	leaf_blocks / k
Index-only	blevel + FF_i * leaf_blocks
	(100-oic)/100 * FF_i * leaf_blocks
Range scan	blevel + FF_i * leaf_blocks + FF_t * clustering_factor
	(100-oic)/100 * FF_i * leaf_blocks + FF_t * clustering_factor

As already pointed out, these adjusted index access cost formulas for oic <> 0 apply only to index accesses within NL joins.

Don't get overly exited over the 0 cost of a unique index lookup in an NL join. It would appear to make NL joins with this kind of index access, e.g. a fully qualified primary key join predicate, cost free. However, 0 and 1 often get special treatment by the cbo and we'll see later what happens to the 0.

Optimizer_Index_Caching

Unlike optimizer_index_cost_adj,

optimizer_index_caching

affects only the cost of NL joins

That means that the optimizer considers the effects of index caching **only** within the same sql, **not** across different sql.

One of the consequences of this is that if you set OIC to your BCHR, as is most often suggested, you are likely overestimating the effect of index caching in the area (NL joins) where the CBO does use it.

The BCHR is made up of 3 components:

a) caching of index root and branch blocks

b) caching of index leaf blocks

c) caching of data blocks

BCHR is likely lifted by very high hit ratios on index root and branch blocks and the hit ratio for index leaf blocks is therefore likely lower than the overall BCHR.

o_i_c_a and o_i_c together

```
Outer table: cost: 7387  cdn: 429  rcz: 23
 Inner table index: LVLS: 2   #LB: 3178  CLUF: 21958
            IX_SEL:  1.0000e+00  TB_SEL:  4.7619e-02
```

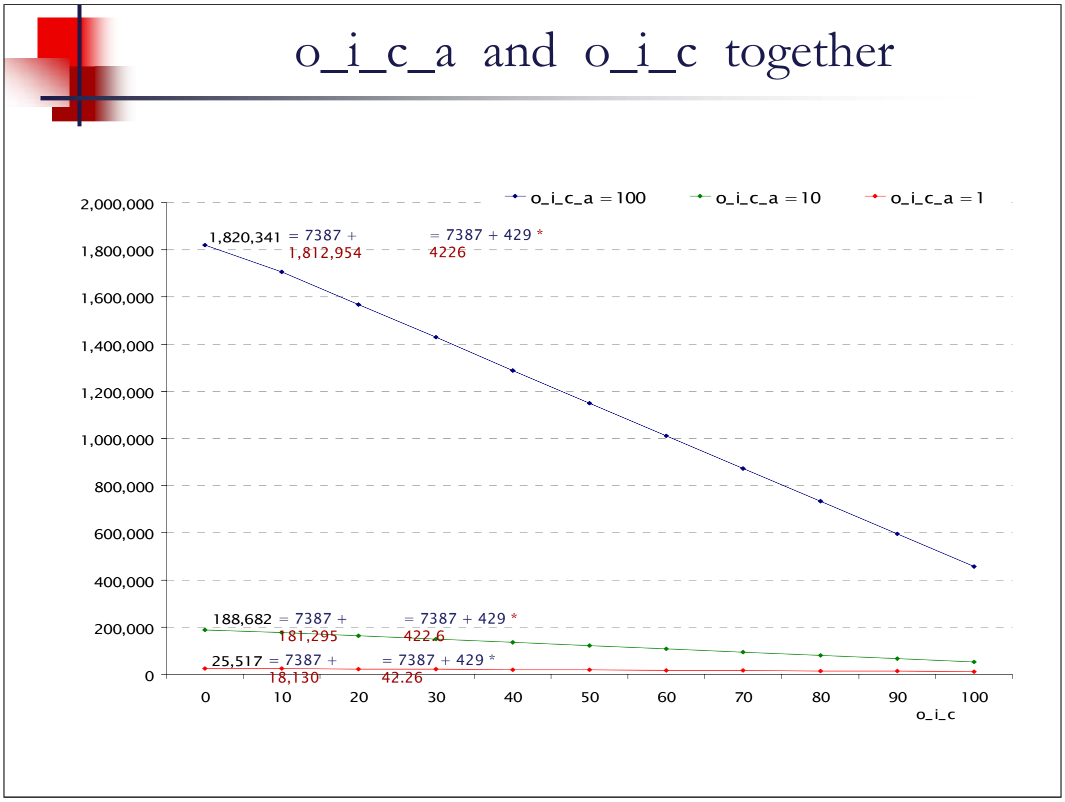
oic	oica:	100	10	1
	rsc_io	join	join	join
0	4,226	1,820,341	188,682	25,517
10	3,959	1,705,798	177,228	24,371
20	3,635	1,566,802	163,328	22,981
30	3,312	1,428,235	149,472	21,595
40	2,988	1,289,239	135,572	20,206
50	2,664	1,150,243	121,673	18,816
60	2,341	1,011,676	107,816	17,430
70	2,017	872,680	93,916	16,040
80	1,694	734,113	80,060	14,654
90	1,370	595,117	66,160	13,264
100	1,046	456,121	52,260	11,874

Having gained an understanding of the individual effects of oica and oic, we look at the combined effect.

It is obvious that the join costs go down with decreasing oica (from left to right) and increasing oic (top down). Not shown is that the rsc_io (index access costs) did not change with decreasing oica. This is akin to the single table access costs where the oica effect was not shown in the trace on the individual index access cost calculations, only in the best_cst value when the access path choice had been made – and was an index access path.

However, the decline doesn't seem to be in direct relation to the oica decrease. oica = 1 is $1/100^{th}$ of oica = 100 but 25,517 is not $1/100^{th}$ of 1,820,341.

On the oic axis, the table shows a residual factor of 1046 for rsc_io even if 100% of the index is considered cached. This factor stems from the table row access: $1046 = 4.7619e^{-2} * 21958$ = tb_sel * cluf



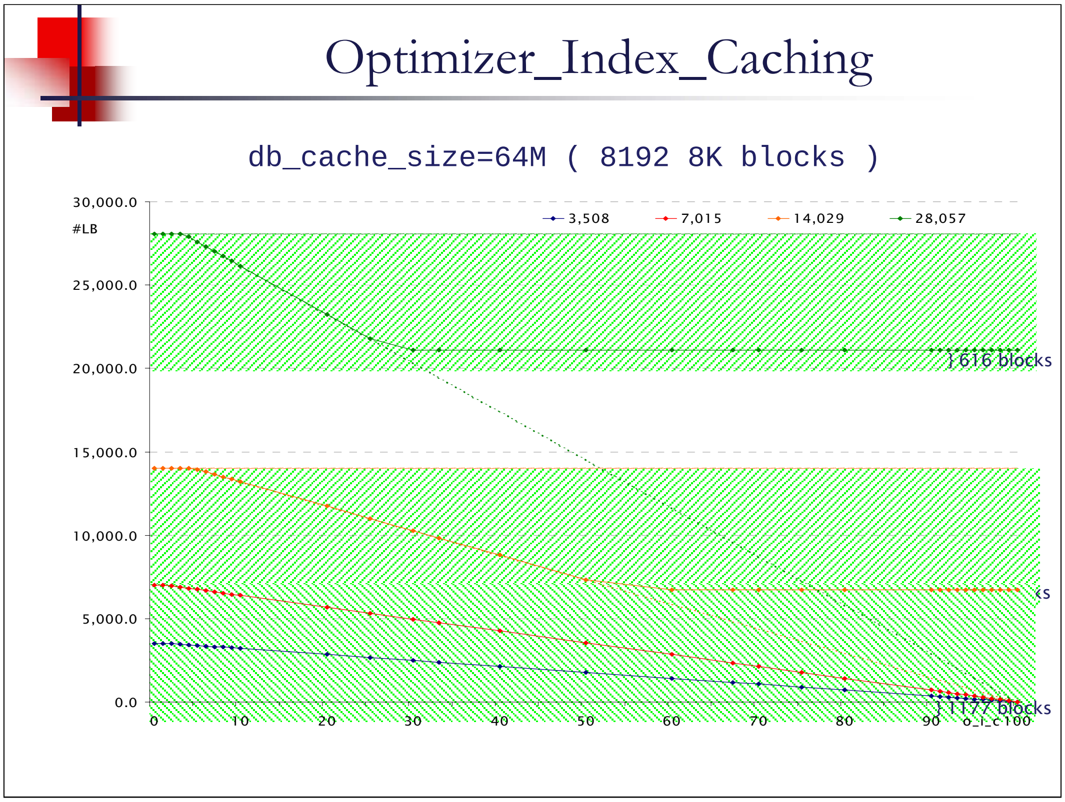
Charting join costs over oic for the three oica values shows again the liner relationship with different slopes for different oica values.

o_i_c_a and o_i_c together

$$\$_{join} = \$_{outer} + oica/100 * cdn * rsc\_io$$

$$rsc\_io \approx \mathit{lvls} + (100 - oic)/100 * ix\_sel * \#lb + tb\_sel * cluf$$

We have already seen both formulas. The first one on slide 9, the second on slide 13. This slide simply puts it all together.

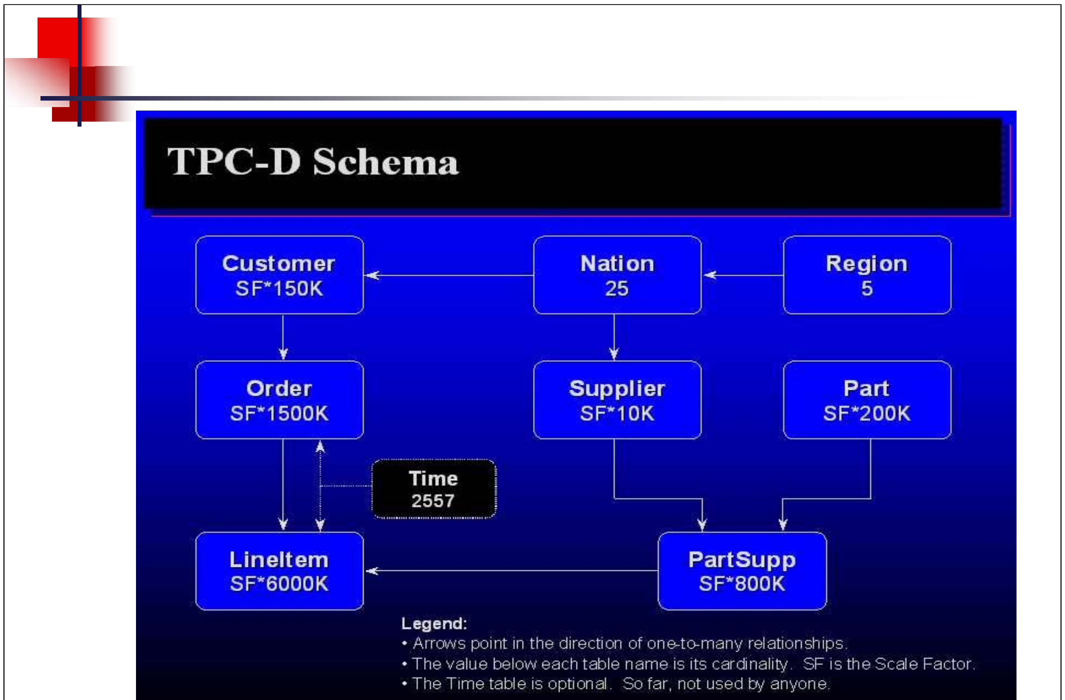


There is a not immediately obvious detail of the oic factor to the index access costs within NL joins. There is a cap, or rather barrier, below which the costs do not fall. This barrier only comes into view when the index size, #LB, exceeds the buffer pool size.

Each of the 4 lines in the chart are the cost declines for indexes of different size – each double that of its predecessor. Otherwise the indexes were identical. The other table in the join was scaled correspondingly.

Two points:

- The larger indexes have a longer flat section (to oic=4) before the cost decrease starts. I mentioned that before
- While the larger indexes begin at a cost slope which would get them to a cost of 0 at oic=100, the cost levels off when the number of purportedly cached index blocks reaches the number of blocks in the pool, minus a cushion. The barrier seems to be reached at ~ oic=27 and oic=54 respectively. 27% of 28,057 (= 54% of 14,029) = 7575.4 which leaves a cushion of 616 blocks in a 8192 block pool.



Source: http://www.tpc.org/information/sessions/sigmod/sld029.htm

For an exploration of plan changes from changing oica nd oic settings I chose the tpc-d benchmark schema. Mainly because there are exact specifications regarding the data and programs available which load with predefined, repeatable data.

tpc-d rather than the better known tpc-c, or any other, benchmark because I came across a dbgen program that can not only load the data specified in the benchmark, but also skewed data (for another project).

Scaling factor for the test was 0.1

The SQL

```
select sum(o.totalprice)
   from order o
   , customer c
   , lineitem l
   , part p
   where o.cust_id = c.cust_id
     and l.order_id = o.order_id
     and l.part_id = p.part_id
     and o.orderpriority = '2-HIGH'
     and c.mktsegment = 'HOUSEHOLD'
     and p.mfgr = 'Manufacturer#2'
     and p.brand = 'Brand#33'
     and l.shipdate > to_date('2005-01-22','yyyy-
   mm-dd')
```

The schema and the data contents are from the tpc-d benchmar.

The sql, however, is not. it was designed such that each of the 4 tables has at least one non-join predicate to give the CBO a choice of an index access path for each single table access path.

Also, the data content for orderpriority was changed from a uniform data distribution of the values

1-URGENT

2-HIGH

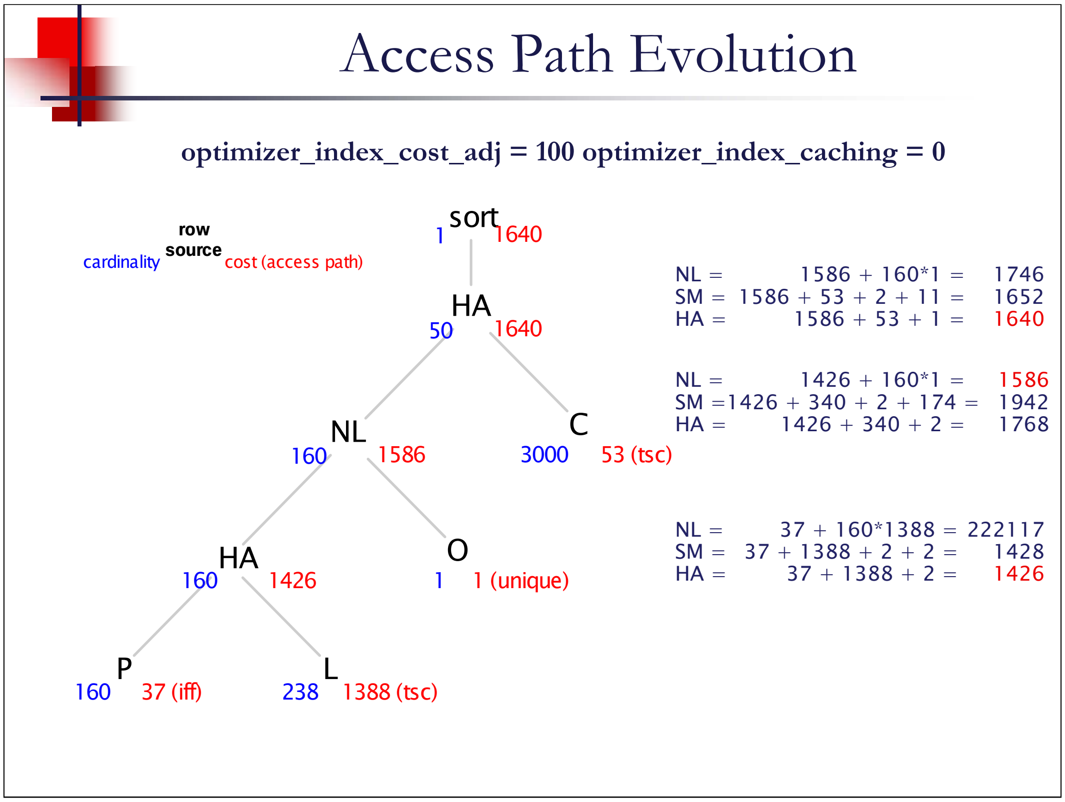
3-MEDIUM

4-NOT SPECIFIED

5-LOW

was changed to a normal distribution with 4-NOT SPECIFIED as the mean and 1-URGENT and 5-LOW as the two extremes.

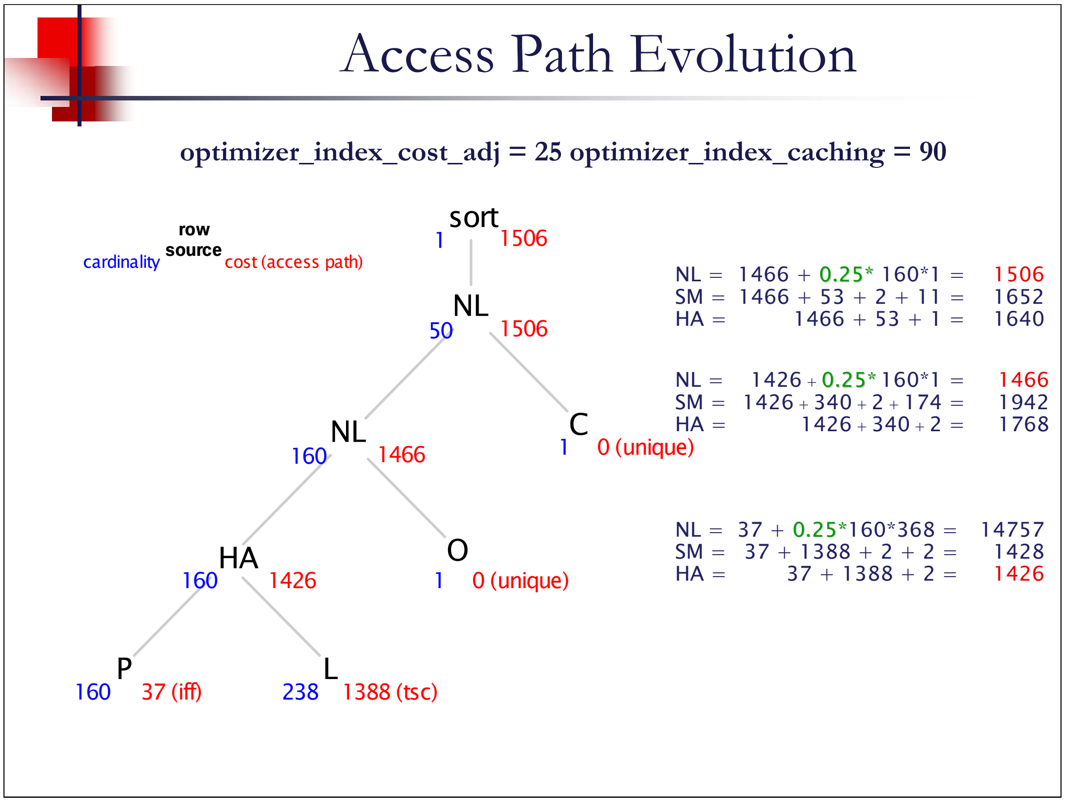
I should note that in the following slides the sole attention was on the plan changes, not which of the resultant plans performs best. In fact, neither plan was actually executed. The sql were only explained with a 10053 trace active.



The base "line"

part has a composite index on SIZE, BRAND, TYPE, PART_ID, MFGR

i.e. both predicates are in the index but not suitable for a range scan. Therefore the iff scan came out the cheapest.



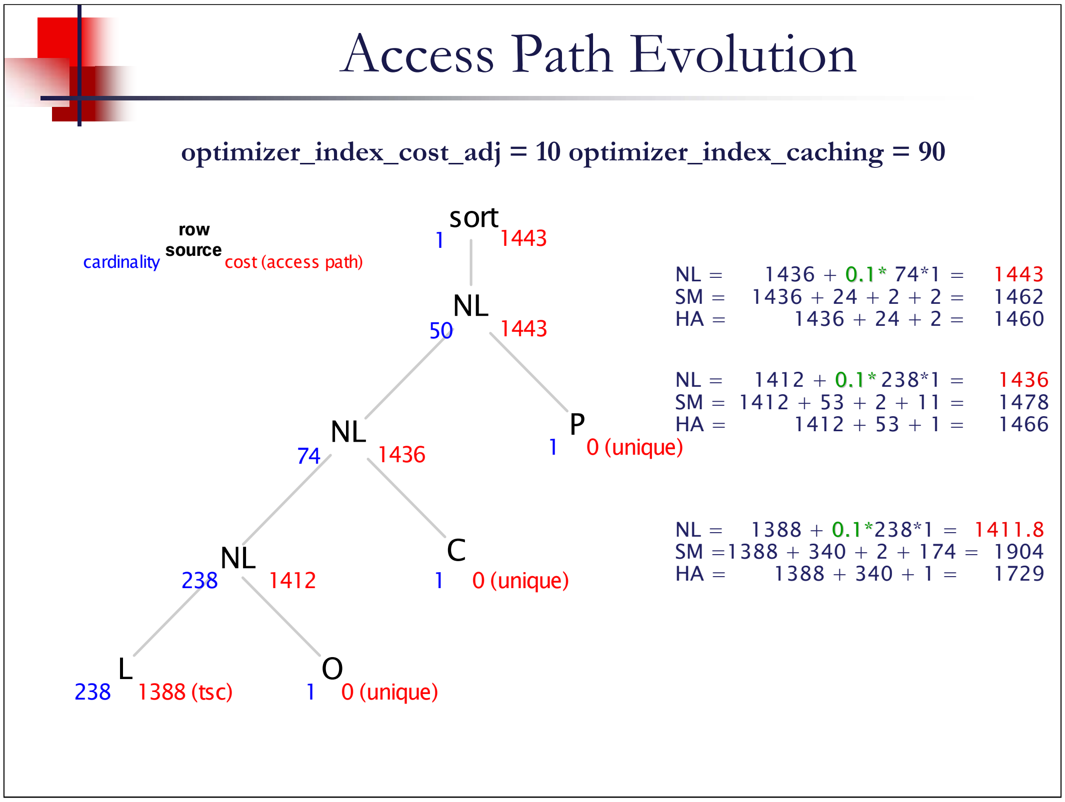
Note that both, oica and oic, were changed here and to values which are in the range most often cited:

10 <= oica <= 40

25 because it makes a nice fraction of 100 (1/4)

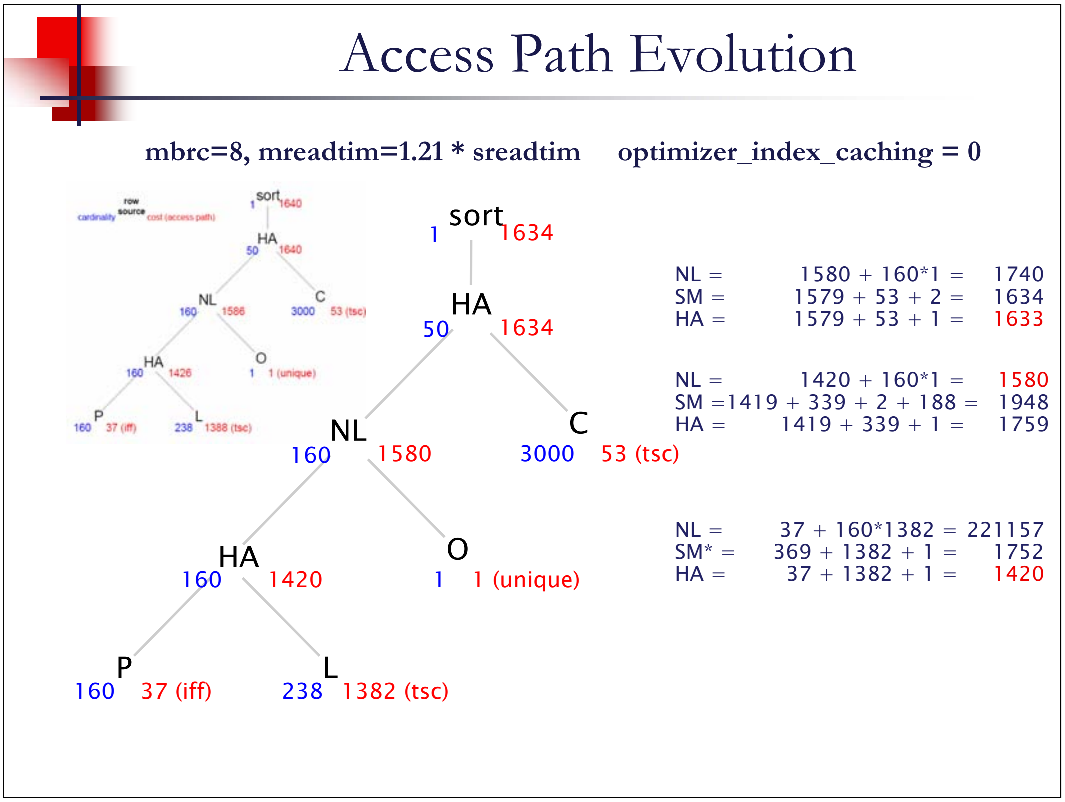
The first join and the cost do not change but the last one does from an HA to a NL.

Note that although individual costs for the index lookups in the NL joins are listed as 0, the NL join is not costed as a freebie but at a cost of 1, discounted by the oica factor.



If we get more aggressive with the oica setting, the plan turns almost upside down, now starting with a full scan of the lineitem table and NL-joining the other tables with unique index-lookup joins.

This confirms that changing these parameters from their defaults values cause the cbo to favor NL joins.

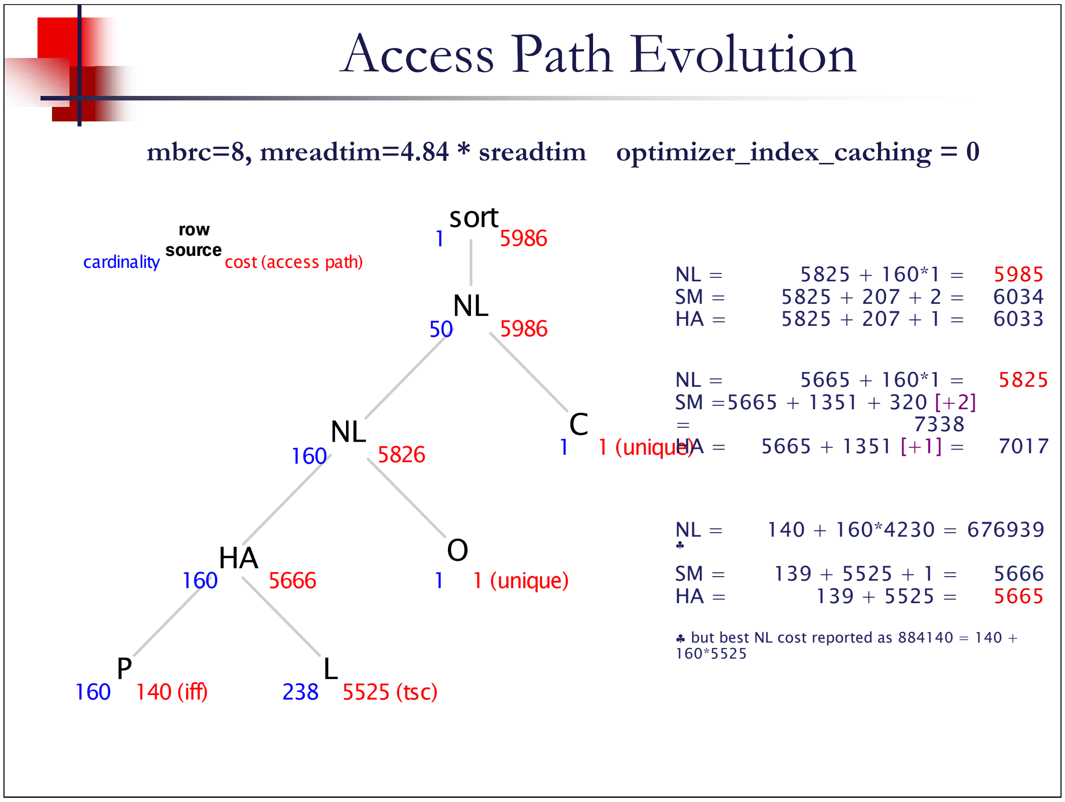


Let us contract the effect of oica with that of system statistics. On the surface, setting oica to 25, ¼ the default, should be equivalent to raising the mreadtim system statistic 4 fold.and setting oica to 10 equivalent to setting mreadtim to 10 times the original. Over the next 3 slides I am going to examine if that is the case.

To avoid the CPU component having an effect, consider
* setting _optimizer_cost_model = IO
* setting the CPUSPEED to something extremely large to minimize the impact.

I chose the latter

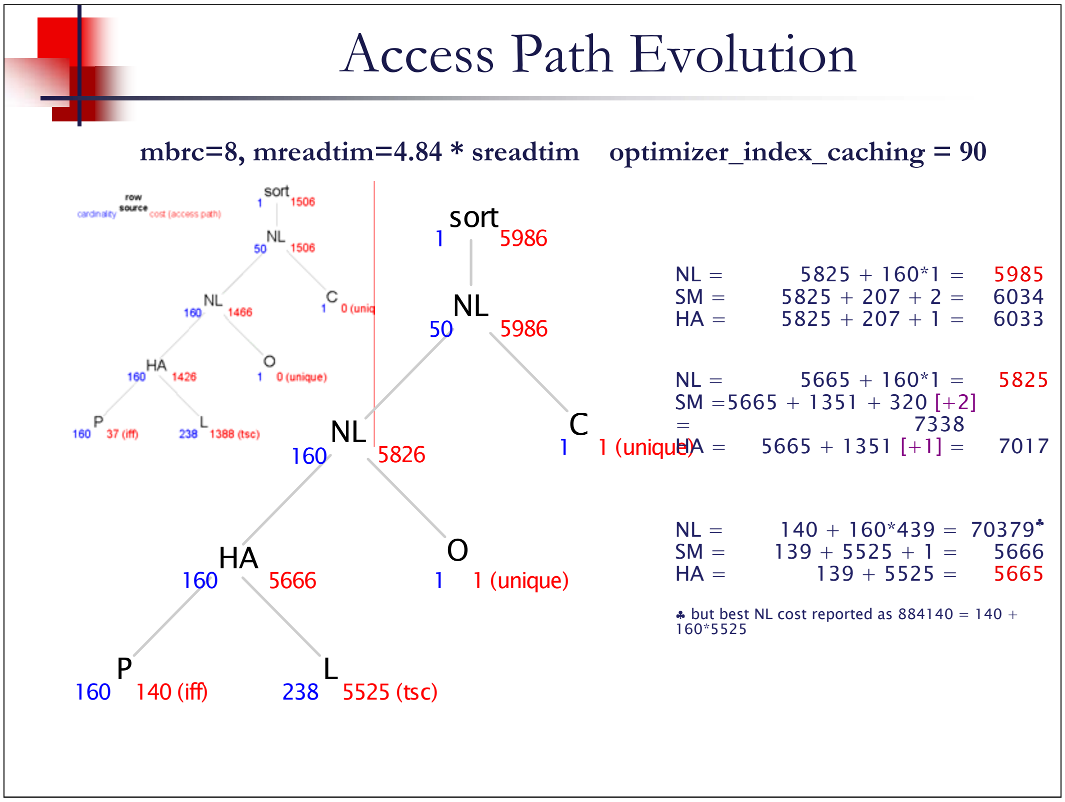
This is the baseline with system statistics. With mbrc=8 and mreadtim=1.21 time sreadtim the cpu costing is equivalent to the old cost model. The component and overall plan costs bear that out.



There is no counterpart to this slide in the prior set with oica/oic. It is here to show the effects of independently raising just mreadtim, analogous to setting oica to 25.

We can see that the Single Table access costs at the bottom left quadrupled from before. This due to the fact that both base plans use scans, iff for P and tsc for L, wich both are directly affected by the higher mreadtim.

Since also the tsc cost for C in the last join also quadrupled, a NL join with index access to C came out cheaper.

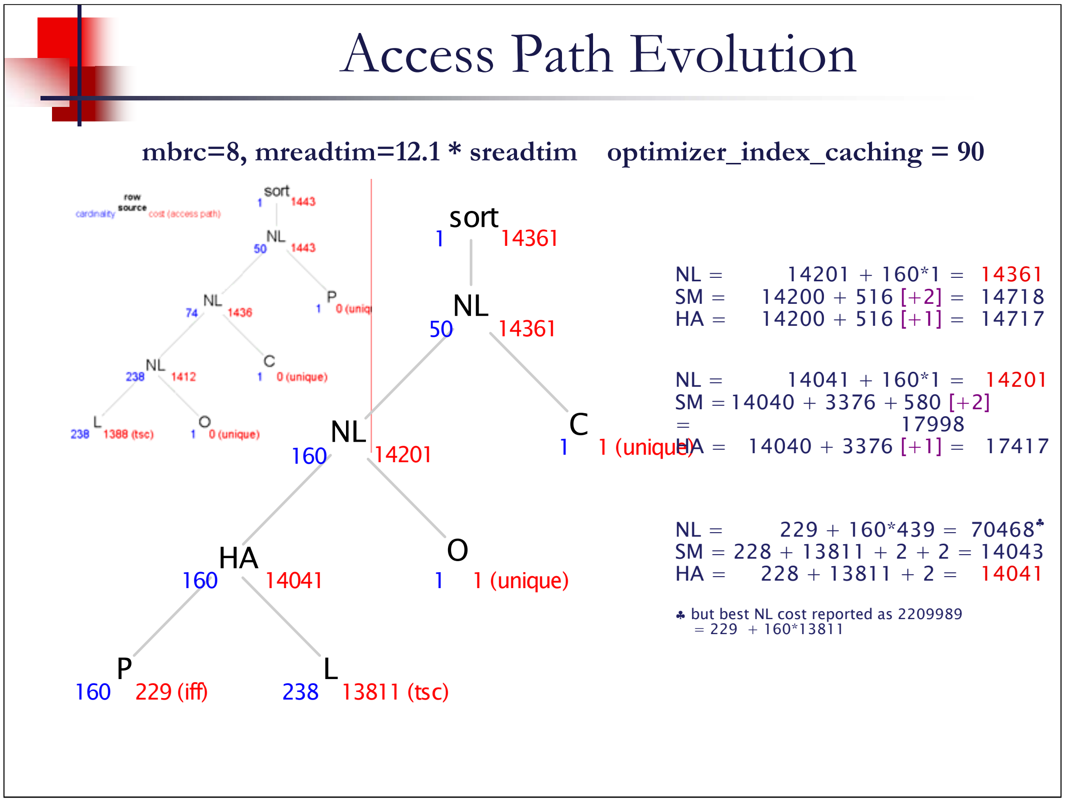


This corresponds to oica=25 and oic=90

The component and overall plan cost are identical to the one with oic=0.

The only difference is in the detail costing of the NL join option between P and L. For unknown reasons, CBO calculates the shown cost for the NL join with index access to L but then lists the "best NL cost" based on a NL join with a tsc for L. It does not matter in this case since either figure is much higher than the HA join.

The corresponding plan with oica/oic is shown in the upper left. The plan is identical but the costs are dramatically different.



Most noteworthy on this slide is that obviously setting mreadtim to n times the base value does not have the same effect as setting oica to 1/n times its base value (100) event though on the surface they appear to do the same thing. However, one is reducing the single-block access costs while the other increases the multi-block access costs. Since other costs remain the same, they gain a greater influence in the oica case as opposed to raising mreadtim where multiblock read costs then dominate and other costs become marginalized.

The two resulting plans are clearly different and it is not just the effect of flattening out the small differences in index access costs which Jonathan Lewis demonstrated so entertainingly with his good_index and bad_index example.

To be honest, I am suspicious about the performance of the "upside-down" access plan of the oica=10, oic=90 option.

My favorite websites

asktom.oracle.com	(Thomas Kyte)
www.evdbt.com	(Tim Gorman)
www.ixora.com.au	(Steve Adams)
www.jlcomp.demon.co.uk	(Jonathan Lewis)
www.hotsos.com	(Cary Millsap)
www.miracleas.dk	(Mogens Nørgaard)
www.oracledba.co.uk	(Connor McDonald)
www.oraperf.com	(Anjo Kolk)
www.orapub.com	(Craig Shallahamer)

Wolfgang Breitling

breitliw@centrexcc.com

Centrex Consulting Corp.

www.centrexcc.com