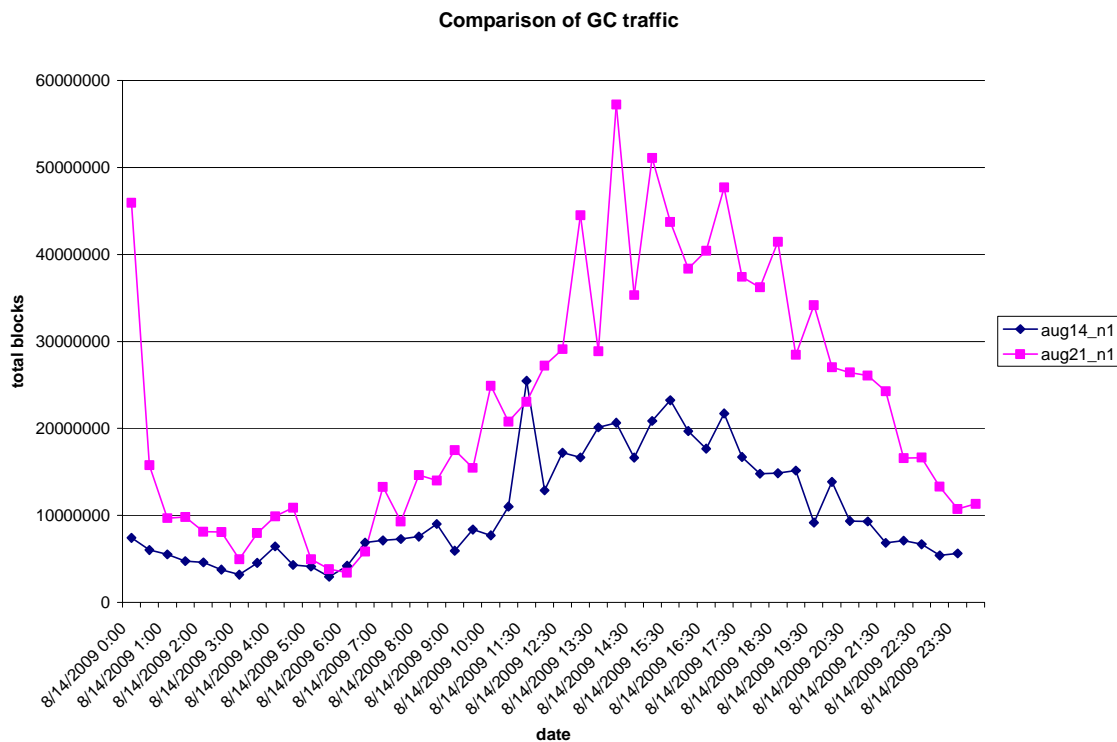


Is plan hash value a final say?

Filed under: [CBO](#), [Performance tuning](#), [RAC](#), [Uncategorized](#) — orainternals @ 12:18 am
Tags: [gc traffic](#), [global cache CR](#), [global cache CUR](#), [oracle performance](#), [plan hash value](#), [scientific tuning](#)

I was reviewing a performance issue with a client recently. Problem is that increased global cache waits causing application slowdown affecting few critical business functions. Using one of my script [gc_traffic.sql](#) and graphing the results with Excel spreadsheet, it is established that there is a marked increase in GC traffic today compared to week earlier. Similar jobs runs every day and so comparing two week days is sufficient to show the GC traffic increase. Graph is between total blocks and AWR snap time in 30 minutes interval. [Click the picture below to review the graph clearly.]



Identifying the object creating this increased GC traffic is essential to identify root cause. We were able to quickly determine that this increase in GC traffic was localized around few SQL statements using ADDM and AWR reports. We decided to focus on one SQL with an obvious increase in elapsed time compared to prior week. So, first question asked, is there a change in the plan? plan_hash_value was reviewed and quickly determined that there is no change in the plan_hash_value.

Little bit of history, there were few changes performed by the vendor over the weekend as a part of few bug fixes. Vendor's argument was that since there is no change to the plan_hash_value, SQL access plan did not change and so, this can't be due to vendor

changes. My Client's argument was that there were no changes to the environment and problem started after the application changes.

There are many different things that can go wrong without changes to the execution plan. We can ignore those conditions for now since (a) there has been no changes to the environment (b) no visible changes to the data (c) no error message and average CR recv time is consistent with prior weeks. Well, Let's cut to the chase. It boiled down to a question "Can SQL plan change without a change in plan_hash_value?". What do you think? Please answer in the pool below.

plan_hash_value and hash_value

Hash_value of a SQL statement is generated from the text of an SQL statement and plan_hash_value is generated from the execution plan of that SQL statement[More accurately, from that child cursors' execution plan and exactly what is involved in generating plan_hash_value is not published]. It is a general belief that plan_hash_value will change even if there is a slightest change in the execution plan. But, that is not always the case!

Test case

We will use a small table to explore this issue.

```
prompt Test case #1: Initial test case with index on columns (n1, n2).
prompt =====
create table t1 (n1 number, n2 number, v1 varchar2(100));
create index t1_n1 on t1(n1,n2);

explain plan for select * from t1 where n1=:b1 and n2=:b2;
select * from table (dbms_xplan.display);
prompt plan hash value in this case is 626762252
```

Plan hash value: 626762252

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	4 (0)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	1		3 (0)	00:00:01

Predicate Information (identified by operation id):

```
-----
2 - access("N1"=TO_NUMBER(:B1) AND "N2"=TO_NUMBER(:B2))
```

So, we got the plan_hash_value as highlighted above. In the following test case we will recreate the index reordering the columns as (n2, n1).

```
prompt Test case #2: Index with re-ordered columns
prompt =====
drop index t1_n1;
```

```
create index t1_n1 on t1(n2,n1);
```

Plan hash value: **626762252**

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	9 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	9 (0)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	5298		1 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - access("N2"=TO_NUMBER(:B2) AND "N1"=TO_NUMBER(:B1))
```

Notice that in the test result above filter predicate changed reflecting index column reordering. But the plan_hash_value did not change. Point is that execution plan can change without a change in plan_hash_value (due to change in the underlying tables).

Let's modify that index dropping a column from the index in the next test case.

prompt Test case #3: dropping a column from index t1_n1.

```
drop index t1_n1;  
create index t1_n1 on t1(n1);
```

```
explain plan for select * from t1 where n1=:b1 and n2=:b2;
```

Plan hash value: 626762252

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	41 (3)	00:00:01
* 1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	41 (3)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	2119		3 (0)	00:00:01

Predicate Information (identified by operation id):

```
1 - filter("N2"=TO_NUMBER(:B2))  
2 - access("N1"=TO_NUMBER(:B1))
```

We see that predicates changed but the plan_hash_value did not change. In the test case below, we will modify index to be a function based index and test this SQL statement. There are also few more self-evident test cases below.

prompt Test case #3: Index is a function based index. Still, no change in the plan_hash_value.

```
create index t1_n1 on t1(to_char(n1));  
explain plan for select * from t1 where to_char(n1)=:b1 and n2=:b2;
```

Plan hash value: 626762252

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	127 (1)	00:00:01
* 1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	127 (1)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	2119		3 (0)	00:00:01

Predicate Information (identified by operation id):

- 1 - filter("N2"=TO_NUMBER(:B2))
- 2 - access(TO_CHAR("N1")=:B1)

prompt Test case #4: Different schema and same SQL. plan_hash_value did not change.

Plan hash value: 626762252

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	78	2 (0)	00:00:01
* 1	TABLE ACCESS BY INDEX ROWID	T1	1	78	2 (0)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	3		1 (0)	00:00:01

Predicate Information (identified by operation id):

- 1 - filter("N2"=TO_NUMBER(:B2))
- 2 - access("N1"=TO_NUMBER(:B1))

plan_hash_value is also case sensitive for index names. In the test cases below, we will create case sensitive indices.

prompt Test case #5: Lower case index_name.. plan_hash_value changed.
 create index "t1_n1" on t1(n1,n2);
 explain plan for select * from t1 where n1=:b1 and n2=:b2;

Plan hash value: 2252949961

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	4 (0)	00:00:01
* 2	INDEX RANGE SCAN	t1_n1	1		3 (0)	00:00:01

prompt Test case #6: Upper case index_name.. plan_hash_value did not change.

create index "T1_N1" on t1(n1,n2);
 explain plan for select * from t1 where n1=:b1 and n2=:b2;
 Plan hash value: 626762252

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	4 (0)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	1		3 (0)	00:00:01

prompt Test case #7: Space in the index name changes plan_hash_value though.

create index "T1_N1 " on t1(n1,n2);

```
explain plan for select * from t1 where n1=:b1 and n2=:b2;
Plan hash value: 1377541522
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		53	4134	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T1	53	4134	4 (0)	00:00:01
* 2	INDEX RANGE SCAN	T1_N1	1		3 (0)	00:00:01

Summary

In summary, `plan_hash_value` is a very good indicator to see if the plan changed or not. Don't get me wrong, I also use `plan_hash_value`, but in addition to comparing CPU time and elapsed time. Execution plan can change even when there is no change to the `plan_hash_value`. Salient points from these test cases are:

1. `Plan_hash_value` is dependent upon partial execution plan not on complete execution plan.
2. If a predicate is moved from `filter_predicate` to `access_predicate` or vice-versa, it doesn't affect `plan_hash_value`.
3. Changes in the parallelism of the queries does not affect `plan_hash_value`. For example, if a query used 4 parallel slaves today and 16 parallel slaves yesterday, that change is not visible through `plan_hash_value`. This behavior is expected as `parallel_adaptive_multiusers` can allocate slaves depending upon the state of that instance.
4. `Plan_hash_value` is case-sensitive to index/table names. Also sensitive to white-space characters.
5. `Plan_hash_value` is not sensitive to index types. For example, if the index type is a function based index as in our test case #4, as long as, index name did not change `plan_hash_value` will remain the same.
6. `Plan_hash_value` is not sensitive to schema either. SQL statement accessing different schemas can have same `plan_hash_value` too.

Back to our problem. One of the index was recreated removing a column and caused optimizer to apply filter predicates at table level increasing number of accesses to table block tremendously, leading to more logical reads, more Global cache waits etc. This problem was amplified since this SQL was executed very frequently and concurrently from all RAC instances.

In my client's defense, this application change was tested thoroughly. But, alas, test data chosen for this performance test was not probing this specific issue. This performance issue did not show up in development. Essentially, chosen data in the performance benchmark suite was not an excellent choice.

As they say in Gaelic language "Go raimh maith agat" to my client for allowing me to post this blog.