

# HASH GROUP BY

## Introduction

So, Here I was merrily enjoying Open World 2010 presentations in SFO, I got a call from a client about a performance issue. Client recently upgraded from Version 9i to Version 10g in an E-Business environment. I had the privilege of consulting before the upgrade, so we setup the environment optimally, and upgrade itself was seamless. Client did not see much regression except one query: That query was running for hours in 10g compared to 15 minutes in 9i.

## Review and Analysis

Reviewed the execution plan in the development database and I did not see any issues with the plan. Execution plan in development and production looked decent enough. I wasn't able to reproduce the issue in the development database either. So, the client allowed me to trace the SQL statement in the production database. Since the size of data in few tables is different between production and development databases, we had to analyze the problem in production environment.

I had to collect as much data possible as the tracing was a one-time thing. I setup a small script to get process stack and process memory area of that UNIX dedicated server process to collect more details, in addition to tracing the process with waits => true.

Execution plan from the production database printed below. [Review the execution plan carefully; it is giving away the problem immediately.] One execution of this statement took 13,445 seconds and almost all of it spent in the CPU time. Why would the process consume 13,719 seconds of CPU time? Same process completed in just 15 minutes in 9i, as confirmed by Statspack reports. [As a side note, we collected enormous amount of performance data in 9i in the Production environment before upgrading to 10g, just so that we can quickly resolve any performance issues, and you should probably follow that guideline too]. That collection came handy and it is clear that SQL statement was completing in 15 minutes in 9i and took nearly 3.75 hours after upgrading the database to version 10g.

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	10	13719.71	13445.94	27	5086407	0	99938
total	12	13719.71	13445.94	27	5086407	0	99938

  

```
24 HASH GROUP BY (cr=4904031 pr=27 pw=0 time=13240600266 us)
24 NESTED LOOPS OUTER (cr=4904031 pr=27 pw=0 time=136204709 us)
24 NESTED LOOPS (cr=4903935 pr=27 pw=0 time=133347961 us)
489983 NESTED LOOPS (cr=3432044 pr=27 pw=0 time=104239982 us)
489983 NESTED LOOPS (cr=2452078 pr=27 pw=0 time=91156653 us)
489983 TABLE ACCESS BY INDEX ROWID HR_LOCATIONS_ALL (cr=1472112 pr=27 pw=0
time=70907109 us)
489983 INDEX RANGE SCAN HR_LOCATIONS_UK2 (cr=981232 pr=0 pw=0 time=54338789
us)(object id 43397)
489983 INDEX UNIQUE SCAN MTL_PARAMETERS_U1 (cr=979966 pr=0 pw=0 time=17972426
us)(object id 37657)
489983 INDEX UNIQUE SCAN HR_ORGANIZATION_UNITS_PK (cr=979966 pr=0 pw=0
time=10876601 us)(object id 43498)
24 INDEX RANGE SCAN UXPP_FA_LOCATIONS_N3 (cr=1471891 pr=0 pw=0 time=27325172
us)(object id 316461)
24 TABLE ACCESS BY INDEX ROWID PER_ALL_PEOPLE_F (cr=96 pr=0 pw=0 time=2191 us)
24 INDEX RANGE SCAN PER_PEOPLE_F_PK (cr=72 pr=0 pw=0 time=1543 us)(object id
44403)
```

## Pstack, pmap and truss

Reviewing pstack output generated from the script shows many function calls kghfrempty, kghfrempty\_ex, qerghFreeHashTable etc, implying hash table operations. Is some operation with hash table consuming time?

```
( Only partial entries shown )
0000000103f41528 kghfrempty
0000000103f466ec kghfrempty_ex
0000000103191f1c qerghFreeHashTable
000000010318e080 qerghFetch
00000001030b1b3c qerstFetch
...
0000000103f41558 kghfrempty
0000000103f466ec kghfrempty_ex
0000000103191f1c qerghFreeHashTable
000000010318e080 qerghFetch
00000001030b1b3c qerstFetch
```

Truss of the process also showed quite a bit of mmap calls. So, the process is allocating more memory to a hash table?

```
...
mmap(0xFFFFFFFF231C0000, 65536, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 7, 0) = 0xFFFFFFFF231C0000
...
pollsys(0xFFFFFFFF7FFF7EC8, 1, 0xFFFFFFFF7FFF7E00, 0x00000000) = 0
mmap(0xFFFFFFFF231D0000, 65536, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED, 7, 0) = 0xFFFFFFFF231D0000
...
```

## Execution plan again

Reviewing the execution plan again showed an interesting issue. I am going to post only two relevant lines from the execution plan below. As you can see that elapsed time at NESTED LOOPS OUTER step is 136 seconds. But the elapsed time at the next HASH GROUP BY step is 13240 seconds, meaning nearly 13,100 seconds spent in the HASH GROUP BY Step alone! Why would the process spend 13,100 seconds in a group by operation? Actual SQL execution took only 136 seconds, but the group by operation took 13,100 seconds. That doesn't make sense, Does it?

```
24 HASH GROUP BY (cr=4904031 pr=27 pw=0 time=13240600266 us)
24 NESTED LOOPS OUTER (cr=4904031 pr=27 pw=0 time=136204709 us)
...
```

## OFE = 9.2.0.8

Knowing that time is spent in the Group by operation and that the 10g new feature Hash Grouping method is in use, I decided to test this SQL statement execution with 9i optimizer. The SQL

completed in 908 seconds with OFE(optimizer\_features\_enabled) set to 9.2.0.8 (data is little bit different since production is an active environment). You can also see that SORT technique is used to group the data.

```
alter session set optimizer_features_enabled=9.2.0.8;
```

```
Explain plan :
call      count          cpu          elapsed          disk          query          current          rows
-----
Parse          1          0.00          0.00           0           0           0           0
Execute        1          0.00          0.00           0           0           0           0
Fetch    106985      887.41        908.25       282379       3344916        158       1604754
-----
total    106987      887.41        908.25       282379       3344916        158       1604754

      4  SORT GROUP BY (cr=2863428 pr=0 pw=0 time=37934456 us)
      4  NESTED LOOPS OUTER (cr=2863428 pr=0 pw=0 time=34902519 us)
      4  NESTED LOOPS (cr=2863412 pr=0 pw=0 time=34198726 us)
286067  NESTED LOOPS (cr=2003916 pr=0 pw=0 time=24285794 us)
286067  NESTED LOOPS (cr=1431782 pr=0 pw=0 time=19288024 us)
286067  TABLE ACCESS BY INDEX ROWID HR_LOCATIONS_ALL (cr=859648 pr=0 pw=0
time=13568456 us)
286067  INDEX RANGE SCAN HR_LOCATIONS_UK2 (cr=572969 pr=0 pw=0 time=9271380
us)(object id 43397)
286067  INDEX UNIQUE SCAN MTL_PARAMETERS_U1 (cr=572134 pr=0 pw=0 time=4663154
us)(object id 37657)
...

```

Knowing the problem is in the GROUP BY step, we setup a profile with `_gby_hash_aggregation_enabled` set to FALSE, disabling the new 10g feature for that SQL statement. With the SQL profile, performance of the SQL statement is comparable to pre-upgrade timing.

This almost sounds like a bug! Bug 8223928 is matching with this stack, but it is the opposite. Well, client will work with the support to get a bug fix for this issue.

## Summary

In summary, you can use scientific methods to debug performance issues. Revealing the details underneath will enable you to come up with a workaround quickly, leading to a faster resolution. Note that, I am not saying hash group by feature is bad. Rather, we seem to have encountered an unfortunate bug which caused performance issues at this client. I think, Hash Grouping is a good feature as the efficiency of grouping operations can be improved if you have ample amount of memory. That's the reason why we disabled this feature at the statement level, NOT at the instance level.