

Oracle 9.2 Event 10046 Segment-level Statistics

Andy Rivenes, AppsDBA Consulting
March 13, 2003

Introduction

In Release 9.2, Oracle has added the ability to track "segment-level statistics". Several v\$ views were added to the data dictionary and segment statistic collection can be enabled selectively to collect these statistics. However, when performing event 10046 tracing these statistics show up as well¹. It appears, and I haven't seen any definitive information on this, that this was introduced with the 9.2.0.2 patch set. Since these are "row source" statistics, they show up as part of the STAT lines. The additional information is part of the row source access operation information, specifically the "op=" field. This paper will explore how this information can be found and how it may be used. There do appear to be some inconsistencies with the data produced, but in general these segment-level statistics appear to give the performance analyst another piece of important information in classifying a transaction's response time and in helping to identify all response time components.

STAT Line Format

In the event 10046 trace file, segment-level statistics show up as part of the STAT lines. These are "row source" statistics, and the additional information is part of the row source access operation information, the "op=" field. The following are two examples, one from a pre-9.2 trace file and one from a 9.2 trace file.

Pre-9.2 trace file:

```
STAT #5 id=1 cnt=1 pid=0 pos=0 obj=43135 op='TABLE ACCESS CLUSTER SEG$ '
```

New 9.2 trace file format:

```
STAT #1 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=167 r=4 w=0 time=31888 us)'
```

Note the new information "(cr=167 r=4 w=0 time=31888 us)". The "cr=" value identifies logical I/O for consistent reads, the "r=" value identifies physical reads, the "w=" value identifies physical writes and the "time=" value identifies the elapsed time and the timing precision (e.g. us – microseconds). Note that "current mode" logical I/O (e.g. cu= in the trace file) is not reported. The segment statistics also appear to report only for EXEC and FETCH operations.

In the context of the STAT lines for a given statement, the values "roll-up" in the hierarchy of the execution plan, at least in most cases². When interpreting the information, a STAT line with a non-zero "obj=" value will identify the statistics for accessing that segment. These values will be "rolled up" through the execution plan, and generally STAT lines with "obj=" values of zero will sum their children's contributing row source statistics.

¹ Since the segment-level statistics show up in any level event 10046 trace, they show up in simple SQL_TRACE traces also.

² We will show an example of a SELECT ... FOR UPDATE where this does not seem to hold true, and there certainly may be other cases as well.

Statistic Presentation

So, what can be done with this information? In pre-9.2 trace files, the following would be a typical summarization of the STAT line information (Release 8.1.7.2):

STATEMENT TEXT

```
select count(*) from dba_users
```

STATEMENT EXECUTION PLAN

Rows	Row	Source	Operation
1			SORT AGGREGATE
12,047			MERGE JOIN
12,048			SORT JOIN
12,047			NESTED LOOPS
12,048			NESTED LOOPS
12,048			NESTED LOOPS
12,048			MERGE JOIN
10			SORT JOIN
9			NESTED LOOPS
10			TABLE ACCESS FULL USER_ASTATUS_MAP
9			TABLE ACCESS BY INDEX ROWID PROFILE\$
162			INDEX RANGE SCAN (object id 43224)
12,056			SORT JOIN
12,047			TABLE ACCESS FULL USER\$
24,094			TABLE ACCESS CLUSTER TS\$
24,094			INDEX UNIQUE SCAN (object id 43126)
24,094			TABLE ACCESS CLUSTER TS\$
24,094			INDEX UNIQUE SCAN (object id 43126)
12,047			TABLE ACCESS BY INDEX ROWID PROFILE\$
216,846			INDEX RANGE SCAN (object id 43224)
12,047			SORT JOIN
7			TABLE ACCESS FULL PROFNAME\$

Now, in 9.2 the information looks like (Release 9.2.0.2):

STATEMENT TEXT

```
select count(*) from dba_users
```

STATEMENT EXECUTION PLAN

Rows	Row	Source	Operation
1			SORT AGGREGATE (cr=167 r=4 w=0 time=31888 us)
36			MERGE JOIN (cr=167 r=4 w=0 time=31863 us)
36			SORT JOIN (cr=164 r=3 w=0 time=30441 us)
36			TABLE ACCESS BY INDEX ROWID PROFILE\$ (cr=164 r=3 w=0 time=29994 us)
649			NESTED LOOPS (cr=163 r=3 w=0 time=25712 us)
36			NESTED LOOPS (cr=161 r=3 w=0 time=22979 us)
36			NESTED LOOPS (cr=87 r=3 w=0 time=21223 us)
36			MERGE JOIN (cr=13 r=3 w=0 time=19489 us)
9			SORT JOIN (cr=6 r=3 w=0 time=18519 us)
9			TABLE ACCESS BY INDEX ROWID PROFILE\$ (cr=6 r=3 w=0 time=18313 us)
163			NESTED LOOPS (cr=5 r=2 w=0 time=11319 us)
9			TABLE ACCESS FULL USER_ASTATUS_MAP (cr=3 r=1 w=0 time=1278 us)
153			INDEX RANGE SCAN I_PROFILE (cr=2 r=1 w=0 time=9769 us) (object id 139)
36			SORT JOIN (cr=7 r=0 w=0 time=837 us)
36			TABLE ACCESS FULL USER\$ (cr=7 r=0 w=0 time=420 us)
36			TABLE ACCESS CLUSTER TS\$ (cr=74 r=0 w=0 time=1535 us)
36			INDEX UNIQUE SCAN I_TS# (cr=2 r=0 w=0 time=206 us) (object id 7)
36			TABLE ACCESS CLUSTER TS\$ (cr=74 r=0 w=0 time=1563 us)
36			INDEX UNIQUE SCAN I_TS# (cr=2 r=0 w=0 time=145 us) (object id 7)
612			INDEX RANGE SCAN I_PROFILE (cr=2 r=0 w=0 time=1219 us) (object id 139)
36			SORT JOIN (cr=3 r=1 w=0 time=1283 us)
1			TABLE ACCESS FULL PROFNAME\$ (cr=3 r=1 w=0 time=936 us)

With this information, we can break down the statistics by segment and summarize each segment's contribution to the resources of the EXEC and FETCH portions of the statement. Based on the above example, the following would be the statistics for each segment:

Object ID	LIO (cr=)	Physical Reads	Physical Writes	Elapsed Time(sec)
94	2	1	0	0.011276
139	4	1	0	0.010988
16	144	0	0	0.002747
280	3	1	0	0.001278
95	3	1	0	0.000936
22	7	0	0	0.000420
7	4	0	0	0.000351
Total	167	4	0	0.027996

This summarization was achieved by using the logic described previously. For each statement's STAT line there will be an id (e.g. id=) associated with that line and a parent id (e.g. pid=) that will refer to that line's parent. The topmost line in the hierarchy will have an id equal to one with a parent id equal to zero. STAT lines with a non-zero "obj=" value will identify the statistics for accessing that segment. These values will be "rolled up" through the execution plan hierarchy, and generally STAT lines with "obj=" values of zero will sum their children's contributing row source statistics.

Statistic Validation

Example 1

If we compare the previous example's object statistic summary with the actual statement level statistics below, we see that the total logical reads for all segments equals the logical reads performed during the EXEC and FETCH operations of the statement execution. We also see the same for the physical reads. This won't always be true, but they should be a subset of those values up to, and possibly equal to, the statement statistic totals.

STATEMENT STATISTICS

Action	Count	CPU	Elapsed	PIO Blks	LIO Blks	Consistent	Current	Rows
PARSE	1	0.090000	0.142649	6	149	149	0	0
EXEC	1	0.000000	0.000465	0	0	0	0	0
FETCH	2	0.020000	0.031909	4	167	167	0	1
Total	4	0.110000	0.175023	10	316	316	0	1

The total elapsed time validation for the segment statistics presents an interesting problem. If we look at the elapsed time for the EXEC and FETCH operations and subtract the CPU time we get a value of 0.012374 seconds, or 12374 us. If we then look at the total elapsed time from our segment statistics we see a value of 0.027996 or 27996 us, but if we go back and look at the total time for all the row source statistics (e.g. time=31888 us) we see that that time is slightly less than our total EXEC and FETCH elapsed times of 0.032374 seconds, or 32374 us. This makes sense if we note that the elapsed time for the segment statistics includes the CPU time necessary to access the row source (e.g. logical I/O) along with any physical I/O time. We should also note that in this case we had no "current mode" I/O, more on that in the next example.

Let's look at a couple of more examples. What happens if we have current mode logical I/O (e.g. cu= in the EXEC or FETCH statements) or if we have both EXEC and FETCH statement statistic values?

Example 2

The following example shows both consistent mode (cr=) and current mode (cu=) I/O during this statement's EXEC operation. In reviewing both the segment statistics and the summary we see that Oracle only reports the consistent mode I/O in the segment statistics. In this example only the EXEC level operations had values, and in the previous example only the FETCH operations had values. It does not appear that segment-level statistics are ever reported for PARSE operation statistics.

STATEMENT TEXT

```
delete from dependency$ where d_obj#=:1
```

STATEMENT STATISTICS

Action	Count	CPU	Elapsed	PIO Blks	LIO Blks	Consistent	Current	Rows
PARSE	6	0.020000	0.012211	1	43	43	0	0
EXEC	6	0.010000	0.019214	3	38	16	22	4
FETCH	0	0.000000	0.000000	0	0	0	0	0
Total	12	0.030000	0.031425	4	81	59	22	4

STATEMENT EXECUTION PLAN

Rows	Row Source	Operation
0	DELETE	(cr=3 r=2 w=0 time=7656 us)
1	TABLE ACCESS BY INDEX ROWID	DEPENDENCY\$ (cr=3 r=0 w=0 time=63 us)
1	INDEX RANGE SCAN	I_DEPENDENCY1 (cr=2 r=0 w=0 time=37 us) (object id 127)

Object ID	LIO (cr=)	Physical Reads	Physical Writes	Elapsed Time(sec)
127	2	0	0	0.000037
96	1	0	0	0.000026
Total	3	0	0	0.000063

Example 3

The third and final example shows that the statistics don't always work as expected. In this example notice that the SQL uses a "FOR UPDATE" clause and in the row source we have a case where the statistics don't roll up as we might expect.

STATEMENT STATISTICS

Action	Count	CPU	Elapsed	PIO Blks	LIO Blks	Consistent	Current	Rows
PARSE	1	0.000000	0.001438	0	0	0	0	0
EXEC	1	0.000000	0.000430	0	4	3	1	0
FETCH	2	0.000000	0.000143	0	3	3	0	1
Total	4	0.000000	0.002011	0	7	6	1	1

STATEMENT TEXT

```
select mynum from andy
where mynum = 99
for update
```

STATEMENT EXECUTION PLAN

Rows	Row Source Operation				Elapsed
1	FOR UPDATE (cr=3 r=0 w=0 time=113 us)				
2	TABLE ACCESS FULL ANDY (cr=6 r=0 w=0 time=276 us)				

Object ID	LIO (cr=)	Physical Reads	Physical Writes	Elapsed Time(sec)
30536	6	0	0	0.000276
Total	6	0	0	0.000276

Other things to note are that we now have consistent read I/O for both the EXEC and FETCH operations. We also have one current mode logical I/O during the EXEC operation that is again ignored from the segment-level statistics.

Conclusion

To conclude, Release 9.2's event 10046 trace segment-level statistics give the performance analyst another piece of important information that helps in identifying the response time components for a database transaction. Hopefully this information will inspire more work and understanding of these statistics and how to apply them to time based optimization.

References

Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2), Oracle Corporation, March 2002, Part No. A96533-01

Oracle9i Database Reference, Release 2 (9.2), Oracle Corporation, March 2002, Part No. A96536-01

Note: 39817.1, *Interpreting Raw SQL_TRACE and DBMS_SUPPORT.START_TRACE output*, Oracle Corporation, 14-AUG-1998