ORACLE 10G RAC AND ASM ON LINUX

PERFORMANCE MEASUREMENTS OF LOW-COST SCALABLE STORAGE

Luca Canali, CERN, March 2006

OVERVIEW

This document details the measurements and findings on storage performance and scalability for production Oracle RAC 10g + ASM at the CERN physics database services. For a description of the Oracle RAC 10g architecture see Ref 1. Performance tests measured with the Orion toolkit on similar hardware are reported in Ref 2.

A configuration with 4 RAC nodes and 4 storage arrays has been used for this test. The measured performance metrics are: I/O operations per second (IOPS) for random small I/O (8KB scattered reads), sequential throughput for full scan operations. Test cases have been built using Oracle's SQL, as detailed in the following, and therefore are considered as directly indicative of the capacity available to real-world database applications (for read-intensive workloads). The key findings are:

Measured maximum throughput for random I/O: 8675 IOPS

Measured maximum throughput for sequential I/O: 745 MB/sec

TEST CONFIGURATION

The measurements described here below have been taken using Oracle RAC on Linux deployed on4-nodes with 2 CPUs each. Oracle's storage manager (ASM) has been used to allocate the database storage over 64 HDs organized in 4 storage arrays and access via a SAN network. Key software and hardware components are:

- 4 dual 3GHz XEON installed with Linux RHEL 3.
- Qlogic QLA2312 HBA
- 4 Fiber Channel storage arrays
- Each storage array (Infortrend) has 16 HDs
- SATA disks spinning at 7200 rpm are used
- 1GB cache RAM per controller (write behind mode)
- No hardware RAID is configured on the array controllers, each HD is mapped directly to the OS as a LUN
- 2Gbps SAN network is used to connect the disk array to the server via a Fiber Channel switch
- Oracle RAC 10g (10.2.0.2) on Linux, 4-node cluster.
- ASM 10.2.0.2 and ASMlib 2.0
- The ASM data disk group used for the test database is built on 64 partitions, mapped to the external half of 64 HDs.
- ASM storage is organized in 2 mirrored failgroups (see also Fig. 2)
- SQL has been used to test and measure the I/O subsystem performance

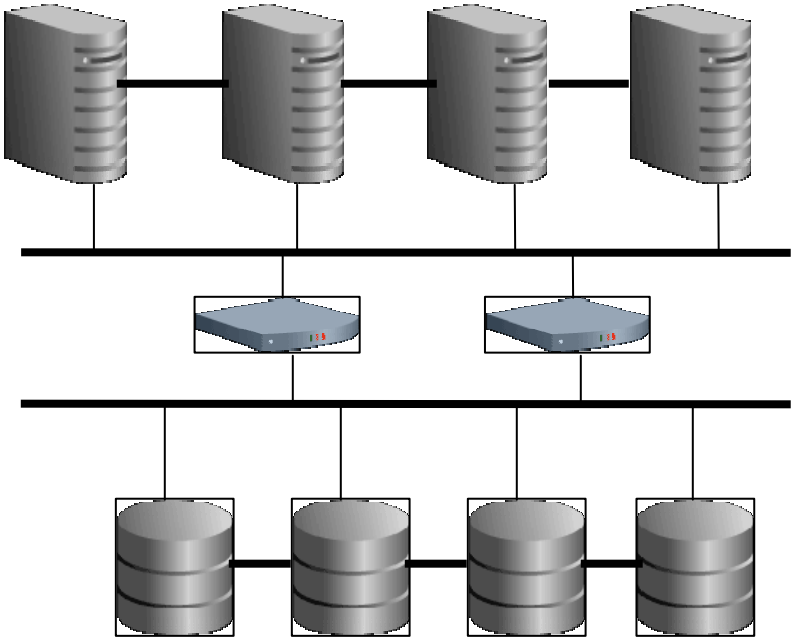


FIGURE 1: *4 NODE ORACLE 10G RAC WITH ASM, ATTACHED TO 4 STORAGE ARRAYS VIA SAN*

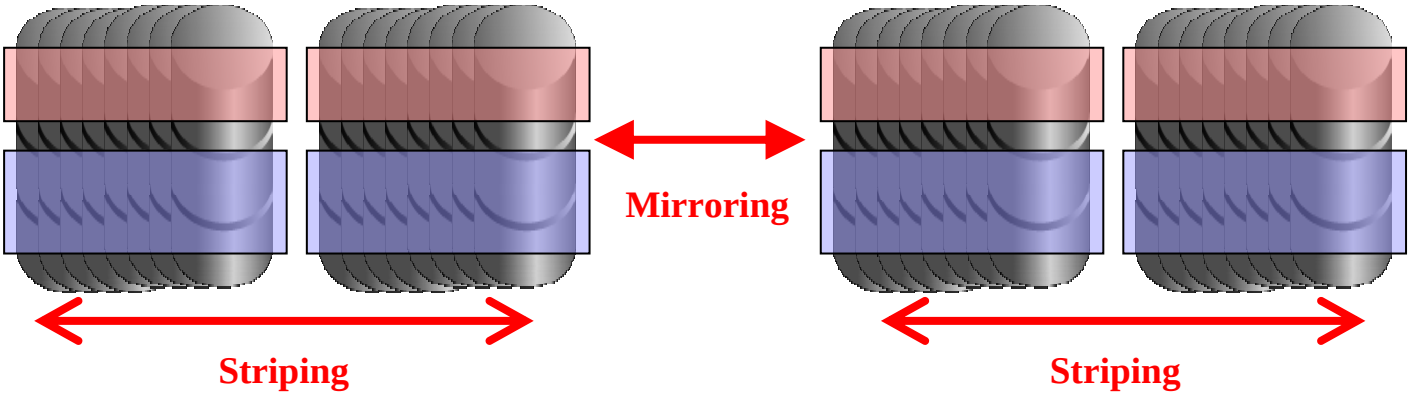
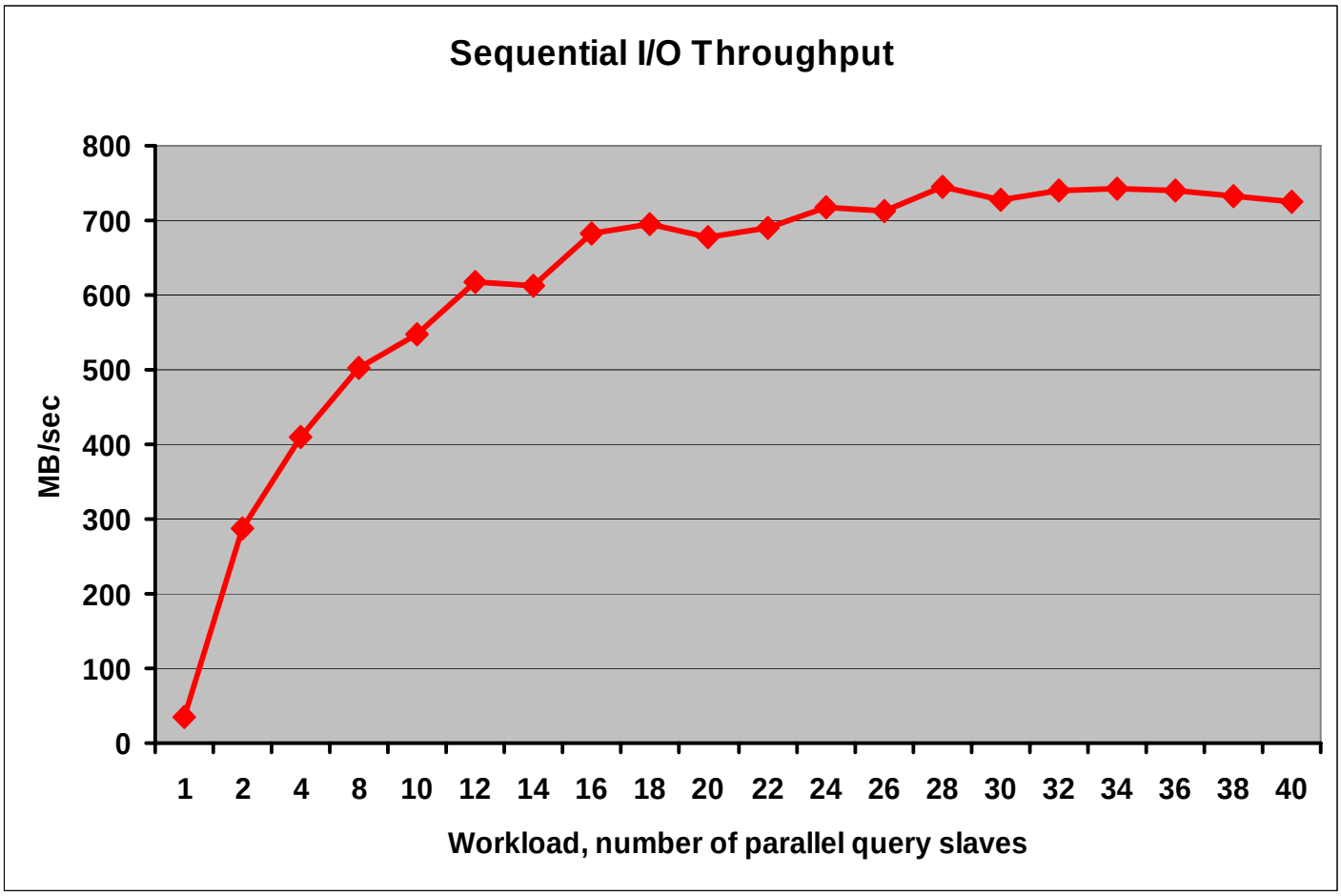


FIGURE 2: *ASM ORGANIZES STORAGE INTO DISKGROUPS, AN IMPLEMENTATION OF THE 'SAME' (STRIPE AND MIRROR EVERYTHING) GUIDELINES. TWO DISKGROUPS ARE SHOW IN THE FIGURE (MARKED WITH PINK AND BLU COLORS). EACH DISKGROUP IS BUILT BY STRIPING AND MIRRORING HD PARTITIONS. EXTERNAL DISK PARTITIONS ARE USED FOR THE DATA DISKGROUP USED IN THE TESTS REPORTED HERE.*

SEQUENTIAL I/O PERFORMANCE

The I/O subsystem has been stressed to capacity for sequential I/O using Oracle's 'parallel query': the parallel full scan operation of a table of 64GB in size. The response time of the full scan operation has been measured as a function of increasing workload, implemented by increasing the degree of parallelism of the test query. The degree of parallelism determines directly the number of slave processes and therefore the number of simultaneous sequential I/O operations outstanding against the I/O subsystem. More details on the test schema and SQL used to gather data reported here are discussed in Appendix A.

The findings for 4cluster nodes and 4 storage arrays (64 disks) are:



SANITY CHECKS – GV$ VIEWS

A series of metrics have been collected during the test as 'sanity checks' and to further validate the findings of this paragraph on the measurement of the maximum sequential throughput. In the table here below the following metrics are reported: Degree of parallelism, Total CPU consumed, Number of blocks read with direct path read access, wait time associated with the direct path read requests. Data shown here below confirm that the workload generated by the test was almost exclusively stressing the I/O subsystem with sequential I/O requests. The data point with 'parallel degree' = 1 and an extended set of statistics and wait events for all the tested degrees of parallelism are reported in Appendix A.

Parallel	Sequental	CPU (s)	physical reads	Wait event: direct

Degree	I/O (MB/s)		direct, blocks	path read (s)
2	287	63	8000000	388
4	410	61	8000000	567
8	504	61	8000000	952
10	549	60	8000000	1095
12	618	60	8000000	1166
14	612	61	8000000	1388
16	683	61	8000000	1427
18	696	63	8000000	1573
20	677	61	8000000	1773
22	690	61	8000000	1944
24	719	62	8000000	2013
26	713	61	8000000	2222
28	745	61	8000000	2300
30	727	61	8000000	2461
32	740	62	8000000	2648
34	742	63	8000000	2761
36	740	63	8000000	2907
38	732	63	8000000	3102
40	724	63	8000000	3370

Sanity check – Throughput measured at the SAN Fabric level

Measurements reported here above show that the 4-node RAC database attached to 4 storage arrays could reach the sequential I/O throughput of about 800 MB/s. This can be explained as saturation of the Fiber Channel controller capacity for the storage arrays. A direct measurement of this fact was done at the SAN switch level using QLogic's SANSurfer tool. The I/O throughput for each port of the SAN switch connected to the storage arrays was measured. A value of about 100 MB/s was obtained for each of the 8 ports connected to storage (each storage array has a dual ported HBA). Appendix D reports a screenshot of the measured I/O throughput for a stress test where I/O saturation at 745 MB/s was measured.

The results found with the SAN performance monitoring confirm the findings discussed above.

Sequential I/O wrap-up

Sequential I/O throughput has been measured from within the database engine using Oracle's parallel query. Oracle RAC with 4 nodes and attached to 4 fiber channel storage arrays have been used for this test. The measured maximum throughput for this configuration is:

Maximum sequential throughput = 745 MB/sec

This measurement is consistent with the expected 800MB/sec saturation value for the FC controllers of the 4 disk arrays used. Moreover this measurement is consistent with the tests performance with Oracle's Orion on similar hardware and reported in Ref 2.

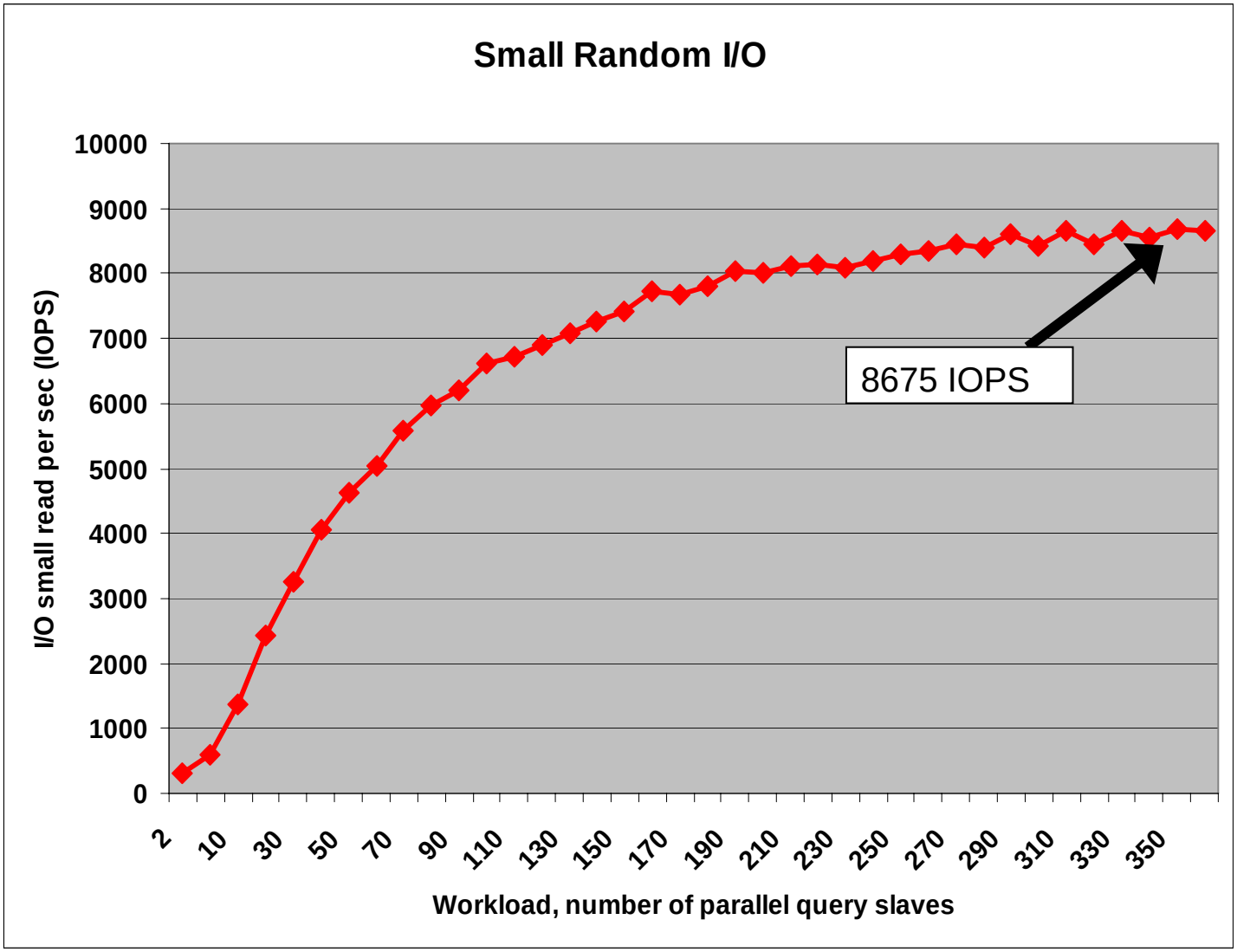
Similar measurement on a configuration with only 1 active node (1 cluster instance up, the other 3 down) showed that the maximum throughput in that case is 398 MB/sec. This number can be explained as being the saturation throughput for the dual ported HBAs installed on the servers (2 Gb per port in this case).

Small Random I/O Performance

The I/O subsystem has been stressed to capacity for small random I/O using Oracle's engine: ad-hoc Oracle's tables and SQL were used to generate millions of small random I/O operations (single block read operations of 8K each). The test

schema consists in a large 'test' table of 2 million blocks (16 GB) and a smaller (100 blocks) 'probe' table containing a list of rowids of the 'test' table arranged in random order. SQL is used to read the large table 'block by block' and randomly, using the probe table. The test tool measures the number of I/O operations per second (IOPS) as a function of the number of simultaneous I/O requests (workload). Details of the test schema and SQL are reported in Appendix B.

The key findings are reported in the graph here below:



Sanity checks – GV$ Views

A series of metrics have been collected during the test as 'sanity checks' and to further validate the findings of this paragraph regarding the measurement of the maximum IOPS for small random IO. In the table here below the following metrics are reported: degree of parallelism, measured IOPS, total CPU consumed, total wait time associated with the 'db file sequential read event', number of blocks read from I/O, number of time the wait event 'db file sequential read event' was issued. Data shown here below confirm that the workload generated by the test was I/O bound and almost exclusively stressed the I/O subsystem with random I/O requests. More data are reported in Appendix A.

Parallel Degree	IOPS	CPU (s)	db file sequential read, wait (s)	physical reads, blocks	db file sequential read, N# waits
2	298	144	13087	2029369	2007128

4	587	140	13192	2083146	2008363
10	1375	121	13963	2007092	2000564
20	2430	139	15400	2004564	2000479
30	3247	143	16988	2004012	2000035
40	4048	155	18553	2008105	2001429
50	4638	161	20153	2004510	2000581
60	5040	168	22124	2004004	2000027
70	5573	172	23540	2004007	2000031
80	5959	180	25277	2009011	2001421
90	6189	179	27201	2004230	2000036
100	6624	191	28982	2004019	2000042
110	6707	186	30872	2005004	2001075
120	6906	188	32692	2004017	2000040
130	7082	192	34402	2004596	2000619
140	7265	194	36073	2004048	2000048
150	7414	200	38203	2009205	2001400
160	7721	196	39760	2004032	2000055
170	7664	206	42145	2006068	2002139
180	7807	198	44102	2004033	2000056
190	8041	200	45274	2004021	2000045
200	8016	201	47713	2004024	2000047
210	8110	201	49471	2004025	2000048
220	8129	214	50949	2009235	2001430
230	8077	213	53311	2004029	2000053
240	8183	222	54225	2018785	2014855
250	8291	212	56915	2004037	2000060
260	8335	207	58803	2006489	2001421
270	8460	217	59652	2009838	2002493
280	8402	210	62511	2004041	2000065
290	8592	211	63104	2004053	2000076
300	8426	214	66842	2004050	2000073
310	8651	212	67435	2004357	2000084
320	8450	222	70104	2006070	2001518
330	8669	215	71176	2004077	2000100
340	8563	217	73880	2005533	2001617
350	8675	217	73577	2004099	2000122
360	8654	219	77003	2004101	2000093

Sanity checks – OS Level

Measurements at the OS level with vmstat and iostat were performed for a test with workload close to saturation. The findings are

- CPU utilization of about 60% (approximately 40% user and 20% system).
- Uniform disk utilization (all 64 disk partitions used for the test showed almost equal IO metrics)
- Average number of read operations per second per disk: 135
- Average IO queue length per disk: 5
- Average utilization per disk: 90%

- Average request size: 8K
- Average time to service IO requests: 30 ms

Sanity check – extended SQL trace

The extended SQL trace (10046 event) has been used to check the execution pattern and in particular the wait events generated by the test tool. The result is that multiple trace files are generated, one per query slave used (see also appendix B for the implementation details). Each trace file contains mainly a long series of wait event lines for the 'db file sequential read' wait event on the large (2M blocks) test table. In total 2 million single blocks reads are performed. Moreover the traces show that the single block read operations probe the test table in random order, that is two contiguous 'wait' lines in the trace files indicate show that different portions of the test table are read.

Note: as discussed in Appendix B, the test table had to be created in the system tablespace for this test. Tests performed using a different tablespace showed that Oracle used an optimization were the test table was read 8 blocks at a time and issued the 'db file scattered read' wait event.

Small Random I/O wrap-up

Small random I/O throughput (maximum IOPS) has been measured using the Oracle database engine. The measurements have been taken using 4 fiber channel storage arrays and Oracle RAC. A 4-node RAC was used, but only one active instance was used for the tests to reduce the overhead of intercluster communication. Ad hoc test SQL has been used to saturate the I/O subsystem with small random I/O read requests. The key findings are

Measured maximum throughput for random I/O: 8675 IOPS

Measured maximum throughput for random I/O per disk: 135 read operations per sec per disk

The findings documented here are consistent with the results of Ref 2, however in that study the maximum IOPS per disk was measured as 100 read operations per sec per disk.

SUMMARY

Performance measurements of the sequential throughput and maximum small random IOPS for a 4 node 10g RAC with 4 SAN-attached storage arrays have been measured. The measurement toolkit has been developed using Oracle's schema and Oracle's query engine. Therefore the tests and results reported here are considered as directly indicative of the capacity available to real-world database applications (with mainly read-only workload) deployed in production.

The key findings are:

Measured maximum throughput for random I/O: 8675 IOPS

Measured maximum throughput for sequential I/O: 745 MB/sec

The RAC database used for these tests is representative of the production database service for physics at CERN. The results shown are a direct consequence of the scalability of RAC, ASM and in general of the deployed DB architecture. In particular the following facts, confirmed from these tests, are particularly relevant for the production databases:

- Read operation using ASM-mirrored disk groups data are load balanced across all disks.
- Oracle ASM distributes evenly database data using the SAME ideas (stripe and mirror everything).
- QLogic HBA multipathing, as configured in production, effectively load balances the I/O workload.

REFERENCES

1. L. Canali, Scalable Oracle 10g Architecture,
https://twiki.cern.ch/twiki/pub/PSSGroup/HAandPerf/Architecture_description.pdf

2. L. Canali, I/O performance tests using Oracle's Orion,
https://twiki.cern.ch/twiki/pub/PSSGroup/HAandPerf/Orion_tests_Dec05.pdf

3. J Loaiza and S Lee, OOW 2005 session 1262,
http://www.oracle.com/technology/deploy/availability/pdf/1262_Loaiza_WP.pdf

APPENDIX A – SEQUENTIAL THROUGHPUT MEASUREMENTS

Sequential throughput has been measured for varying workload using Oracle's SQL. Parallel full table scan of a large table of 8 million blocks (64 GB) is used to generate sequential I/O requests against the storage. Since ASM is used, Oracle tablespace data is organized on the disks in mirrored stripes (allocation units/extents) of 1MB. Therefore sequential I/O for this tests can also be considered as large random I/O with 1MB chunk size (the same as for the Orion tool).

TEST SCHEMA TABLE DEFINITION:

```
Create bigfile tablespace testio datafile '+DATADG' size 70G;

create table testseq1 pctused 10 pctfree 80
tablespace testio nologging
as
select rownum num, rpad('name',2000,'P') name
from all_objects a, all_objects b
where rownum<=8000000;
exec dbms_stats.gather_table_stats('TESTUSER','TESTSEQ1')
```

KEY INSTANCE PARAMETERS:

```
db_bock_size=8192
sga_target=0
db_cache_size=200m
shared_pool_size=300m
large_pool_size=300m
parallel_max_servers=256
parallel_min_servers=256
parallel_adaptive_multi_user=false
pga_aggregate_target=3g
parallel_execution_message_size=4096
```

TEST SQL:

```
select /*+ parallel (a PUT_HERE_PARAL_DEGREE) */ count(*) from SYSTEM.TESTSEQ1 a;
```

Execution Plan:

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name     | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |          |     1 |  8033   (0)| 00:01:37 |
|   1 |  SORT AGGREGATE          |          |     1 |            |          |
|   2 |   PX COORDINATOR         |          |       |            |          |
|   3 |    PX SEND QC (RANDOM)   | :TQ10000 |     1 |            |          |
|   4 |     SORT AGGREGATE       |          |     1 |            |          |
|   5 |      PX BLOCK ITERATOR   |          |  7991K|  8033   (0)| 00:01:37 |
|   6 |       TABLE ACCESS FULL| TESTSEQ1 |  7991K|  8033   (0)| 00:01:37 |
-------------------------------------------------------------------------------
```

Test Measurements:

In the text above a few metrics from gv$sysstat and gv$system_event have been reported, more data can be found here below. The PL/SQL text of the test program can be seen in Appendix C. Each metric value reported here is an average of two measurements.

From gv$sysstat:

We can see that most of the execution time is spent waiting for I/O, and little time is spent in CPU or other wait events. The I/O issued by the query is direct path IO, as normal for parallel query. The case Parall # =1 is an exception.

Parallel Degree	Sequential I/O (MB/s)	Elapsed (s)	CPU (s)	physical reads	physical reads direct	physical writes	Wait user IO (s)	Wait appl (s)	Wait concur (s)	Wait cluster, (s)
1	35	1877	167	7994874	0	4918	1703	0	2	3
2	287	228	63	8000000	8000000	0	388	0	0	0
4	410	160	61	8000000	8000000	34	567	0	0	0
8	504	130	61	8000000	8000000	0	952	0	0	0
10	549	119	60	8000000	8000000	48	1095	0	0	0
12	618	106	60	8000002.5	8000000	0	1166	0	0	0
14	612	107	61	8000000	8000000	914	1388	0	0	0
16	683	96	61	8000000	8000000	2	1427	0	0	0
18	696	94	63	8000269	8000000	76	1574	3	1	1
20	677	97	61	8000000	8000000	0	1773	0	0	0
22	690	95	61	8000000	8000000	574	1944	0	0	0
24	719	91	62	8000000	8000000	0	2013	0	0	0
26	713	92	61	8000000	8000000	0	2222	0	0	0
28	745	88	61	8000000	8000000	30	2300	0	0	0
30	727	90	61	8000000	8000000	0	2461	0	0	0
32	740	89	62	8000000	8000000	11	2648	0	0	0

34	742	88	63	8000000	8000000	18	2761	0	0	0
36	740	89	63	8000000	8000000	4	2907	0	0	0
38	732	89	63	8000000	8000000	10	3102	0	0	0
40	724	90	63	8000000	8000000	56	3370	0	0	0

FROM GV$SYSTEM_EVENT:

Direct path read is main event, with the exception of Parall# =1. We can see that the total amount of time spent waiting for I/O increases with the parallelism, because more parallel slaves are used (the DB trades throughput against latency for parallel queries).

Parallel Degree	db file sequential read, sec	db file sequential read, N#	db file scattered read, sec	db file scattered read, N#	direct path read, sec	direct path read, N#
1	1	338	1702	186630	0	0
2	0	0	0	0	388	187364
4	0	0	0	0	567	187556
8	0	0	0	0	952	187792
10	0	0	0	0	1095	187792
12	0	3	0	0	1166	187792
14	0	0	0	0	1388	187784
16	0	0	0	0	1427	187792
18	1	200	0	26	1573	187784
20	0	0	0	0	1773	187792
22	0	0	0	0	1944	187783
24	0	0	0	0	2013	187777
26	0	0	0	0	2222	187777
28	0	0	0	0	2300	187768
30	0	0	0	0	2461	187784
32	0	0	0	0	2648	187792
34	0	0	0	0	2761	187758
36	0	0	0	0	2907	187758
38	0	0	0	0	3102	187761
40	0	0	0	0	3370	187792

APPENDIX B – SMALL RANDOM I/O MEASUREMENTS

The I/O subsystem has been stressed to capacity for sequential I/O using Oracle's engine. The test schema consists in a large 'test' table of 2 million blocks (16 GB) and a much smaller (100 blocks) 'probe' table, which contained a list of rowids of the 'test' table arranged in random order. SQL is used to probe the large table 'block by block' in random order. Nested loop join is used to implement this probe-test mechanism. Increasing workload is implemented by using multiple query 'slaves', that is parallel query is used to read the probe table.

Test schema table definition:

```
Create table testtable1 pctused 10 pctfree 80
tablespace system nologging
as
select rownum num, rpad('name',2000,'P') name
from all_objects a, all_objects b
where rownum<=2000000;
exec dbms_stats.gather_table_stats('SYSTEM','TESTTABLE1')
```

Note: the system tablespace must be used for this test to ensure that the test table is probed by single blocks reads by the SQL described below. When a different tablespace is used an optimization is used by the Oracle engine, where I/O request of 8 blocks are issued against the test table instead of single block reads.

Key instance parameters:

```
db_bock_size=8192
sga_target=0
db_cache_size=200m
shared_pool_size=300m
large_pool_size=300m
parallel_max_servers=370
parallel_min_servers=370
parallel_adaptive_multi_user=false
pga_aggregate_target=3g
parallel_execution_message_size=4096
```

Test SQL:

```
select /*+ USE_NL(t p) parallel(p PUT_HERE_PARAL_DEGREE)*/
 sum(1)
from testtable1 t, probetest1 p
where
 t.rowid=p.id
;
```

Execution Plan:

```
-------------------------------------------------------------------------------------------
| Id  | Operation                      | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |            |     1 |    22 | 13154   (1)| 00:02:38 |
|   1 |  SORT AGGREGATE                |            |     1 |    22 |            |          |
|   2 |   PX COORDINATOR               |            |       |       |            |          |
|   3 |    PX SEND QC (RANDOM)         | :TQ10000   |     1 |    22 |            |          |
|   4 |     SORT AGGREGATE             |            |     1 |    22 |            |          |
|   5 |      NESTED LOOPS              |            |  1989K|    41M| 13154   (1)| 00:02:38 |
|   6 |       PX BLOCK ITERATOR        |            |       |       |            |          |
|   7 |        TABLE ACCESS FULL       | PROBETEST1 |  2010K|    19M|     7   (0)| 00:00:01 |
|   8 |       TABLE ACCESS BY USER ROWID| TESTTABLE1 |     1 |    12 |     1   (0)| 00:00:01 |
```

TEST MEASUREMENTS:

In the text above a few metrics from gv$sysstat and gv$system_event have been reported, more data can be found here below. The PL/SQL text of the test program can be seen in Appendix C. Each metric value reported here is an average of two measurements.

FROM GV$SYSSTAT:

The test reported here for small random I/O was done using only 1 cluster node, to reduce the overhead of the cluster communication. Therefore the gv$ metrics are equivalent to v$ in this case. The bottleneck of the test workload is access to the I/O subsystem via single block read requests (see also wait events in the next table). CPU usage for this test is also notable and reaches about 50% for measurements performed with the highest parallelism. Note: the RAC node used for these tests has 2 CPUs.

Parallel Degree	Sequential I/O (MB/s)	Elapsed (s)	CPU (s)	physical reads	physical reads direct	physical writes	Wait user IO (s)	Wait appl (s)	Wait concur (s)	Wait cluster, (s)
2	298	6719	144	2029369	5073	19106	13092	4	0	0
4	587	3410	140	2083146	4603	13193	13217	1	0	0
10	1375	1455	121	2007092	4134	4110	13966	0	0	0
20	2430	823	139	2004564	4134	1110	15403	0	0	0
30	3247	616	143	2004012	3977	121	16991	0	0	0
40	4048	494	155	2008105	4290	5553	18558	0	0	0
50	4638	431	161	2004510	3977	336	20159	0	0	0
60	5040	397	168	2004004	3977	179	22131	0	0	0
70	5573	359	172	2004007	3977	132	23550	0	1	0
80	5959	336	180	2009011	4074	5516	25292	0	2	0
90	6189	323	179	2004230	4194	293	27215	0	2	0
100	6624	302	191	2004019	3977	166	28997	0	3	0
110	6707	298	186	2005004	3977	478	30894	0	6	0
120	6906	290	188	2004017	3977	206	32716	0	9	0
130	7082	282	192	2004596	3977	188	34426	0	9	0
140	7265	275	194	2004048	3977	239	36102	0	12	0

150	7414	270	200	2009205	4290	5658	38232	0	30	0
160	7721	259	196	2004032	3977	254	39787	0	17	0
170	7664	261	206	2006068	3977	574	42180	0	46	0
180	7807	256	198	2004033	3977	355	44133	0	23	0
190	8041	249	200	2004021	3977	272	45306	0	28	0
200	8016	250	201	2004024	3977	272	47749	0	37	0
210	8110	247	201	2004025	3977	272	49509	0	48	0
220	8129	246	214	2009235	4290	5777	50989	0	131	0
230	8077	248	213	2004029	3977	291	53350	0	180	0
240	8183	244	222	2018785	3977	935	54272	0	271	0
250	8291	241	212	2004037	3977	369	56961	0	157	0
260	8335	240	207	2006489	4134	1896	58891	0	138	0
270	8460	236	217	2009838	4034	6026	59712	0	225	0
280	8402	238	210	2004041	3977	558	62572	0	231	0
290	8592	233	211	2004053	3977	364	63154	0	317	0
300	8426	237	214	2004050	3977	371	66895	0	183	0
310	8651	231	212	2004357	3977	408	67494	0	221	0
320	8450	237	222	2006070	4290	5882	70168	0	409	0
330	8669	231	215	2004077	3977	449	71249	0	300	0
340	8563	234	217	2005533	3977	529	73949	0	362	0
350	8675	231	217	2004099	3977	1415	73651	0	957	0
360	8654	231	219	2004101	3977	435	77083	4	687	0

FROM GV$SYSTEM_EVENT:

I/O waits and in particular the wait event ‘db file sequential read’ are the bottleneck of the system is the I/O subsystem, as intended. The number of times the ‘db file sequential read’ wait event is equal to the number of physical reads (see table above). This is another confirmation that I/O workload generated by the test was single block reads of the test table (2 million blocks).

Parall #	db file sequential read, sec	db file sequential read, N#	db file scattered read, sec	db file scattered read, N#	direct path read, sec	direct path read, N#
2	13087	2007128	5	1203	0	1214
4	13192	2008363	24	3249	1	769
10	13963	2000564	1	114	1	288
20	15400	2000479	0	2	3	408
30	16988	2000035	0	0	3	364
40	18553	2001429	1	107	5	813
50	20153	2000581	0	2	6	571
60	22124	2000027	0	0	8	665
70	23540	2000031	0	0	9	798

80	25277	2001421	2	132	12	1094
90	27201	2000036	0	0	14	1214
100	28982	2000042	0	0	15	997
110	30872	2001075	0	2	22	1327
120	32692	2000040	0	0	23	1327
130	34402	2000619	0	0	25	1327
140	36073	2000048	0	6	29	1327
150	38203	2001400	3	140	26	1640
160	39760	2000055	0	0	26	3977
170	42145	2002139	0	2	34	3977
180	44102	2000056	0	0	31	3977
190	45274	2000045	0	0	32	3977
200	47713	2000047	0	0	36	3977
210	49471	2000048	0	0	39	3977
220	50949	2001430	5	139	35	4290
230	53311	2000053	0	0	39	3977
240	54225	2014855	0	3	47	3977
250	56915	2000060	0	0	46	3977
260	58803	2001421	4	85	84	4134
270	59652	2002493	4	132	56	4034
280	62511	2000065	0	0	61	3977
290	63104	2000076	0	0	50	3977
300	66842	2000073	0	0	54	3977
310	67435	2000084	1	21	58	3977
320	70104	2001518	1	28	63	4290
330	71176	2000100	0	0	73	3977
340	73880	2001617	0	0	69	3977
350	73577	2000122	0	0	73	3977
360	77003	2000093	0	6	80	3977

APPENDIX C – TEST PROGRAM

The following is the test program used to measure the small random I/O performance measurements reported in this paper. It is implemented as a stored procedure with one input parameter, the degree parallelism (number of simultaneous I.O requests). The test procedure used to measure sequential I/O is completely analogous.

```
create or replace procedure testrandomio (paralldeg_in in number)
as
 v_time1 timestamp;
 v_DeltaT INTERVAL DAY TO SECOND;
 v_DeltaSec number;
 v_dummy1 number;
 v_query varchar2(300);
 v_tab varchar(1) default chr(9);
 v_cpu1 number;
 v_cpu2 number;
 v_dbtime1 number;
 v_dbtime2 number;
 v_sortdisk1 number;
 v_sortdisk2 number;
 v_phywrite1 number;
 v_phywrite2 number;
 v_phyread1 number;
 v_phyread2 number;
 v_phyreaddir1 number;
 v_phyreaddir2 number;
 v_phyreadbytes1 number;
 v_phyreadbytes2 number;
 v_wait_userio1 number;
 v_wait_userio2 number;
 v_wait_app1 number;
 v_wait_app2 number;
 v_waitconcurr1 number;
 v_waitconcurr2 number;
 v_wait_clust1 number;
 v_wait_clust2 number;
 v_db_file_sequ1 number;
 v_db_file_sequ2 number;
 v_db_file_scatt1 number;
 v_db_file_scatt2 number;
 v_parseela1 number;
 v_parseela2 number;
 v_waits_sequ1 number;
 v_waits_sequ2 number;
 v_wait_time_sequ1 number;
 v_wait_time_sequ2 number;
 v_waits_scat1 number;
```

```
 v_waits_scat2 number;
 v_wait_time_scat1 number;
 v_wait_time_scat2 number;
 v_waits_direct1 number;
 v_waits_direct2 number;
 v_wait_time_direct1 number;
 v_wait_time_direct2 number;
begin

for i in 1..2 loop
 select
  sum(case event when 'db file sequential read' then TOTAL_WAITS end) waits_sequ,
  sum(case event when 'db file sequential read' then TIME_WAITED end) wait_time_sequ,
  sum(case event when 'db file scattered read' then TOTAL_WAITS end) waits_scat,
  sum(case event when 'db file scattered read' then TIME_WAITED end) wait_time_scat,
  sum(case event when 'direct path read' then TOTAL_WAITS end) waits_direct,
  sum(case event when 'direct path read' then TIME_WAITED end) wait_time_direct
 into v_waits_sequ1, v_wait_time_sequ1, v_waits_scat1, v_wait_time_scat1,
      v_waits_direct1, v_wait_time_direct1
 from
  gv$system_event;

 select
  sum(case name when 'CPU used by this session' then value end) CPU,
  sum(case name when 'DB time' then value end) DBTime,
  sum(case name when 'sorts (disk)' then value end) SortDisk,
  sum(case name when 'physical writes' then value end) Phywrite,
  sum(case name when 'physical reads' then value end) PhyRead,
  sum(case name when 'physical reads direct' then value end) PhyReadDir,
  sum(case name when 'physical read bytes' then value end) PhyReadBytes,
  sum(case name when 'parse time elapsed' then value end) parseela,
  sum(case name when 'user I/O wait time' then value end) Wait_UserIO,
  sum(case name when 'application wait time' then value end) Wait_App,
  sum(case name when 'concurrency wait time' then value end) Wait_concurr,
  sum(case name when 'cluster wait time' then value end) Wait_Clust
 into
  v_cpu1, v_dbtime1, v_sortdisk1, v_phywrite1, v_phyread1, v_phyreaddir1,
  v_phyreadbytes1, v_parseela1, v_wait_userio1, v_wait_app1, v_waitconcurr1, v_wait_clust1
 from
  gv$sysstat;

-- PLACE YOUR QUERY HERE!
 v_time1:=systimestamp;
 v_query:='select /*+USE_NL(t p) parallel(p '||paralldeg_in
          ||')*/  sum(1) from system.testtable1 t, system.probetest1 p '
          ||'where t.rowid=p.id';

 execute immediate v_query into v_dummy1;
```

```
 v_DeltaT:=systimestamp-v_time1;
 v_DeltaSec:=extract(hour from v_DeltaT)*3600+extract(minute from v_DeltaT)*60+extract(second from
v_DeltaT);
-- END QUERY

 select
  sum(case name when 'CPU used by this session' then value end) CPU,
  sum(case name when 'DB time' then value end) DBTime,
  sum(case name when 'sorts (disk)' then value end) SortDisk,
  sum(case name when 'physical writes' then value end) Phywrite,
  sum(case name when 'physical reads' then value end) PhyRead,
  sum(case name when 'physical reads direct' then value end) PhyReadDir,
  sum(case name when 'physical read bytes' then value end) PhyReadBytes,
  sum(case name when 'parse time elapsed' then value end) parseela,
  sum(case name when 'user I/O wait time' then value end) Wait_UserIO,
  sum(case name when 'application wait time' then value end) Wait_App,
  sum(case name when 'concurrency wait time' then value end) Wait_concurr,
  sum(case name when 'cluster wait time' then value end) Wait_Clust
 into
  v_cpu2, v_dbtime2, v_sortdisk2, v_phywrite2, v_phyread2, v_phyreaddir2,
  v_phyreadbytes2, v_parseela2, v_wait_userio2, v_wait_app2, v_waitconcurr2, v_wait_clust2
 from
  gv$sysstat;

 select
  sum(case event when 'db file sequential read' then TOTAL_WAITS end) waits_sequ,
  sum(case event when 'db file sequential read' then TIME_WAITED end) wait_time_sequ,
  sum(case event when 'db file scattered read' then TOTAL_WAITS end) waits_scat,
  sum(case event when 'db file scattered read' then TIME_WAITED end) wait_time_scat,
  sum(case event when 'direct path read' then TOTAL_WAITS end) waits_direct,
  sum(case event when 'direct path read' then TIME_WAITED end) wait_time_direct
 into v_waits_sequ2, v_wait_time_sequ2, v_waits_scat2, v_wait_time_scat2,
      v_waits_direct2, v_wait_time_direct2
 from
  gv$system_event;

 dbms_output.put(paralldeg_in||v_tab);
 dbms_output.put(v_DeltaT||v_tab);
 dbms_output.put(v_DeltaSec||v_tab);
 dbms_output.put(v_dummy1||v_tab);
 dbms_output.put(v_cpu2-v_cpu1||v_tab);
 dbms_output.put(v_dbtime2-v_dbtime1||v_tab);
 dbms_output.put(v_sortdisk2-v_sortdisk1||v_tab);
 dbms_output.put(v_phywrite2-v_phywrite1||v_tab);
 dbms_output.put(v_phyread2-v_phyread1||v_tab);
 dbms_output.put(v_phyreaddir2-v_phyreaddir1||v_tab);
 dbms_output.put(v_phyreadbytes2-v_phyreadbytes1||v_tab);
 dbms_output.put(v_parseela2-v_parseela1||v_tab);
 dbms_output.put(v_wait_userio2-v_wait_userio1||v_tab);
```

```
 dbms_output.put(v_wait_app2-v_wait_app1||v_tab);
 dbms_output.put(v_waitconcurr2-v_waitconcurr1||v_tab);
 dbms_output.put(v_wait_clust2-v_wait_clust1||v_tab);
 dbms_output.put(v_wait_time_sequ2-v_wait_time_sequ1||v_tab);
 dbms_output.put(v_waits_sequ2-v_waits_sequ1||v_tab);
 dbms_output.put(v_wait_time_scat2-v_wait_time_scat1||v_tab);
 dbms_output.put(v_waits_scat2-v_waits_scat1||v_tab);
 dbms_output.put(v_waits_direct2-v_waits_direct1||v_tab);
 dbms_output.put(v_wait_time_direct2-v_wait_time_direct1||v_tab);

 dbms_output.put_line('');

 dbms_lock.sleep(1);
end loop;

end;
/
```

APPENDIX D– I/O THROUGHPUT MEASURED ON THE FC FABRIC

QLogic SANsurfer is a GUI that allows configuration and monitoring of the Fiber Channel infrastructure. The following screenshot shows the throughput measured on the QLogic switch during the test described in this paper. The 8 graphs shown correspond to the 8 ports used by the 4 RAC nodes. Each RAC node has a dual ported HBA, where port 1 is connected to SAN switch 1 and port 2 to switch 2. The SAN switches are then connected to the storage arrays. The graphs below show for each HBA port a throughput of about 100MB/sec (during the test window, zero before and after the test). This is in agreement with the measured sequential I/O throughput discussed above in this document.