ORION: ORACLE I/O NUMBERS CALIBRATION TOOL

Orion is currently only available as beta, unsupported software. Oracle will accept no liability for problems arising from its use, nor does Oracle warrant its fitness for any purpose. Its use is purely discretionary at the sole risk of the user.

OVERVIEW

Orion is a tool for predicting the performance of an Oracle database without having to install Oracle or create a database. Unlike other I/O calibration tools, Orion is expressly designed for simulating Oracle database I/O workloads using the same I/O software stack as Oracle. It can also simulate the effect of striping performed by ASM. The following types of I/O workloads are supported:

1. Small Random I/O. OLTP applications typically generate random reads and writes whose size is equivalent to the database block size, typically 8 KB. Such applications typically care about the throughput in I/Os Per Second (IOPS) and about the average latency (I/O turn-around time) per request. These parameters translate to the transaction rate and transaction turn-around time at the application layer.

   Orion can simulate a random I/O workload with a given percentage of reads vs. writes, a given I/O size, and a given number of outstanding I/Os. The I/Os are distributed across all disks.

2. Large Sequential I/O. Data warehousing applications, data loads, backups, and restores generate sequential read and write streams composed of multiple outstanding 1 MB I/Os. Such applications are processing large amounts of data, like a whole table or a whole database and they typically care about the overall data throughput in MegaBytes Per Second (MBPS).

   Orion can simulate a given number of sequential read or write streams of a given I/O size with a given number of outstanding I/Os. Orion can optionally simulate ASM striping when testing sequential streams.

3. Large Random I/O. A sequential stream typically accesses the disks concurrently with other database traffic. With striping, a sequential stream is spread across many disks. Consequently, at the disk level, multiple sequential streams are seen as random 1 MB I/Os, which we also call Multi-User Sequential I/O.

4. Mixed Workloads. Orion can simulate 2 simultaneous workloads: Small Random I/O and either Large Sequential I/O or Large Random I/O. This enables you to simulate, for example, an OLTP workload of 8 KB random reads and writes with a backup workload of 4 sequential read streams of 1 MB I/Os.

For each type of workload, Orion can run tests at different levels of I/O load to measure performance metrics like MBPS, IOPS and I/O latency. Load is expressed in terms of the number of outstanding asynchronous I/Os. Internally, for each such ***load level***, the Orion software keeps issuing I/O requests as fast as they complete to maintain the I/O load at that level. For random workloads (large and small), the load level is the number of outstanding I/Os. For large sequential workloads, the load level is a combination of the number of sequential streams and the number of outstanding I/Os per stream. Testing a given workload at a range of load levels helps the user understand how performance is affected by load.

TEST TARGETS

Orion can, in theory, be used to test any disk-based character device that supports asynchronous I/O. It has been tested on the following types of targets:

1. DAS (direct-attached) storage: Orion can be used to test the performance of one or more local disks or

volumes on the local host.

2. SAN (storage-area network) storage: Orion can be run on any host that has all or parts of the SAN storage mapped as character devices. The devices can correspond to striped or un-striped volumes exported by the storage array(s) or individual disks.
3. Orion has not been extensively tested on NAS (network-attached storage), although it may successfully run on NAS on certain operating systems platforms. In general, the performance results on NAS storage are dependent on the I/O patterns with which the data files have been created and updated. Therefore, you should initialize the data files appropriately before running Orion.

Orion for Storage Vendors

Storage vendors can use Orion to understand how Oracle databases will perform on their storage arrays. Vendors can also use Orion to formulate a recommendation on how to best configure their storage arrays for Oracle databases.

Orion for Oracle Administrators

Oracle administrators can use Orion to evaluate and compare different storage arrays, based on their expected workloads. They can also use Orion to determine the optimal number of network connections, storage arrays, storage array controllers, and disks for their peak workloads. Appendix A describes ways of characterizing a database's current I/O workload to infer its IOPS and MBPS requirements.

Getting Started With Orion

1. Download Orion. Select the version of Orion that corresponds to the operating system of your host server. Orion is currently available for Linux/x86, Solaris/SPARC and Windows/x86.
2. Install Orion:
   a. Linux/Solaris: Unzip the Orion binary into a directory of your choice and add it to your PATH, if preferred.
      ```
      $ gunzip orion10.2_linux.gz
      ```
   b. Windows: Run the Installer, it will let you choose the directory where Orion and its DLLs will be installed.
      ```
      C:\temp> orion10.2_windows.msi
      ```
3. Select a test name, a unique identifier for this calibration run. We will use the example test name "mytest" through the rest of this document.
4. Create a file named mytest.lun. In the file, list the raw volumes or files to test. Put one volume name per line. Do not put comments or anything else in this file. For example:

   ```
   /dev/raw/raw1
   /dev/raw/raw2
   /dev/raw/raw3
   /dev/raw/raw4
   /dev/raw/raw5
   /dev/raw/raw6
   /dev/raw/raw7
   /dev/raw/raw8
   ```

5. Verify that all volumes are accessible using "dd" or other equivalent data copy tools. A typical sanity-check is to do something like this (example: Linux):

```
$ dd  if=/dev/raw/raw1 of=/dev/null bs=32k count=1024
```

Depending on your platform, the data copy tool used and its interface may be different from above.

6. Verify that your platform has the necessary libraries installed to do asynchronous I/Os. The Orion test is completely dependent on asynchronous I/O. On Linux and Solaris, the library libaio needs to be in one of the standard lib directories or accessible through the shell environment's library path variable (usually LD_LIBRARY_PATH or LIBPATH, depending on your shell). Windows has built-in asynchronous I/O libraries, so this issue doesn't apply.
7. As a first test, we suggest a "simple" test run. The simple test measures the performance of *small random reads* at different loads and then (separately) *large random reads* at different loads. These results should give an idea as to how the I/O performance differs with I/O type and load. Such a test can be done with this short command line:

```
$ ./orion -run simple -testname mytest -num_disks 8
ORION: ORacle IO Numbers -- Version 10.2.0.1.0
Test will take approximately 30 minutes
Larger caches may take longer
```

The I/O load levels generated by Orion take into account the number of disk spindles being tested (provided by you in num_disks). Keep in mind that the number of spindles *may or may not be* related to the number of volumes specified in the mytest.lun file, depending on how these volumes are mapped.

8. The results of these tests will be recorded in the output files shown below. The file mytest_summary.txt is a good starting point for verifying your input parameters and analyzing the output. The files mytest_*.csv contain comma-separated values for several I/O performance measures (discussed in detail below).

Output Files

Orion creates several output files. Continuing the "mytest" example, these files are explained below:

1. **mytest_summary.txt**: This file contains:
    a. Input parameters
    b. Maximum throughput observed for the Large Random/Sequential workload
    c. Maximum I/O rate observed for the Small Random workload
    d. Minimum latency observed for the Small Random workload
2. **mytest_mbps.csv:** Comma-separated value file containing the data transfer rate (MBPS) results for the Large Random/Sequential workload. This and all other CSV files contain a two-dimensional table. Each row in the table corresponds to a large I/O load level and each column corresponds to a small I/O load level. Thus, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (for random large I/O tests) or the number of sequential streams (for sequential large I/O tests).

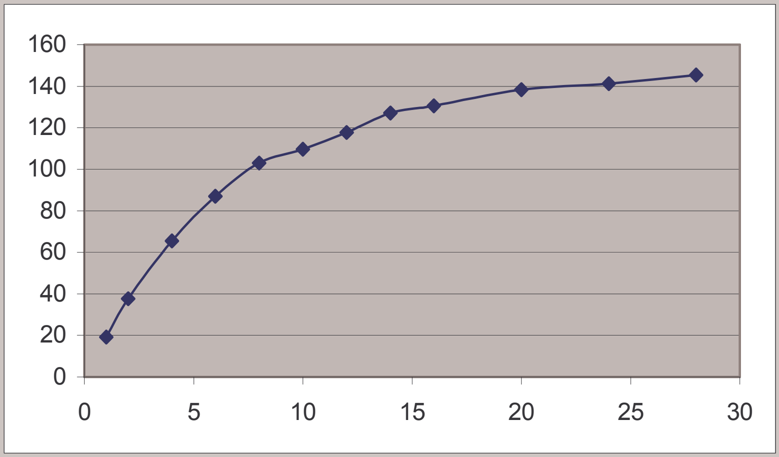
    The first few data points of the Orion MBPS output CSV file for "mytest" are shown below. The simple mytest command-line given earlier does not test combinations of large and small I/Os. Hence, the MBPS file has just one column corresponding to 0 outstanding small I/Os. In the example below, at a load level of 8 outstanding large reads and no small I/Os, we get a data throughput of 103.06 MBPS.

```
Large/Small,      0
          1,  19.18
          2,  37.59
          4,  65.53
          6,  87.03
          8, 103.06
```

```
10,   109.67
. .    . . .
. .    . . .
```

The following chart illustrates the data transfer rate measured at different large I/O load levels. This chart can be generated by loading mytest_mbps.csv into Microsoft Excel or OpenOffice and graphing the data points using the application. Orion does not directly generate such graphs. The x-axis corresponds to the number of outstanding large reads and the y-axis corresponds to the throughput observed.

Large	MBPS
1	19.18
2	37.59
4	65.53
6	87.03
8	103.06
10	109.67
12	117.7
14	127.06
16	130.55
20	138.42
24	141.21
28	145.5



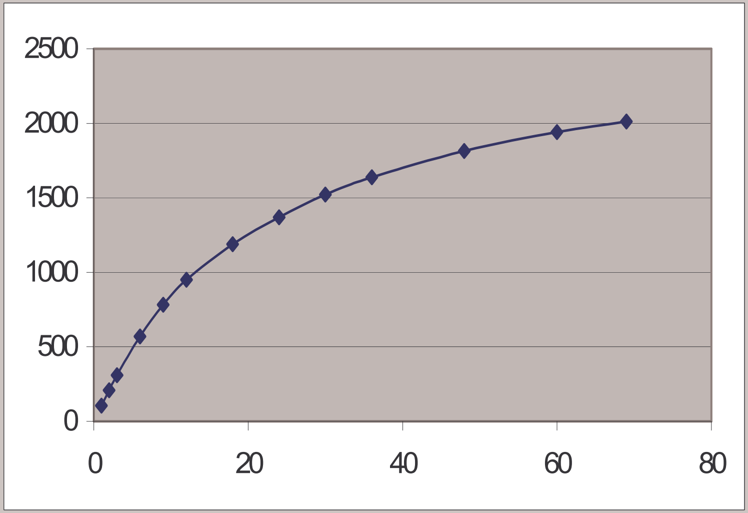
3. **mytest_iops.csv:** Comma-separated value file containing the I/O throughput (in IOPS) results for the Small Random workload. Like in the MBPS file, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (when testing large random) or the number of sequential streams (for large sequential).

   In the general case, this CSV file contains a two-dimensional table. However, for the simple mytest command-line given earlier, we aren't testing combinations of large and small I/Os. Hence, the IOPS results file just has one row with 0 large I/Os. In the example below, with 12 outstanding small reads and no large I/Os, we get a throughput of 951 IOPS.

```
Large/Small,      1,     2,     3,     6,     9,    12 . . . .
0,              105,   208,   309,   569,   782,   951 . . . .
```

The graph below (generated by loading mytest_iops.csv into Excel and charting the data) illustrates the IOPS throughput seen at different small I/O load levels.

Small	IOPS
1	105
2	208
3	309
6	569
9	782
12	951
18	1189
24	1369
30	1521
36	1638
48	1815
60	1941
69	2013



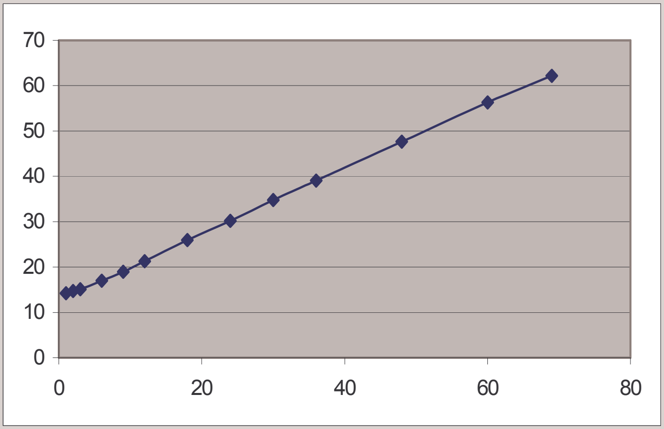
4. **mytest_lat.csv:** Comma-separated value file containing the latency results for the Small Random workload. Like in the MBPS and IOPS files, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (when testing large random I/Os) or the number of sequential streams.

   In the general case, this CSV file contains a two-dimensional table. However, for the simple mytest command-line given earlier, we don't test combinations of large and small I/Os. Hence, the IOPS results file just has one row with 0 large I/Os, as shown below. In the example below, at a sustained load level of 12 outstanding small reads and no large I/Os, we get an I/O turn-around latency of milliseconds.

```
Large/Small,     1,      2,      3,      6,      9,     12 . . . .
0,           14.22, 14.69, 15.09, 16.98, 18.91, 21.25 . . . .
```

   The following graph (generated by loading mytest_lat.csv into Excel and charting the data) illustrates the small I/O latency at different small I/O load levels for mytest.

1	14.22
2	14.69
3	15.09
6	16.98
9	18.91
12	21.25
18	25.92
24	30.2
30	34.8
36	39.04
48	47.68
60	56.3
69	62.16



5. **mytest_trace.txt:** Contains extended, unprocessed test output.

Except for mytest_trace.txt, if any of these files already exist, they will be overwritten. Each run of Orion appends to mytest_trace.txt (or the equivalent trace file name for your test name).

NOTE: IF ANY ERROR OCCURS DURING THE TEST, IT WILL BE PRINTED TO STANDARD OUTPUT.

INPUT PARAMETERS

Orion can be used to test any of the workloads described in the overview using its various command-line options. In this section, we describe these options and the output files.

MANDATORY INPUT PARAMETERS

run: Test run level. This option provides simple command lines at the simple and normal run levels and allows complex commands to be specified at the advanced level. If not set as `-run advanced`, then setting any other non-mandatory parameter (besides `-cache_size` or `-verbose`) will result in an error.

> **simple:** Generates the Small Random I/O and the Large Random I/O workloads for a range of load levels. In this option, small and large I/Os are tested in isolation. The only optional parameters that can be specified at this run level are -cache_size and –verbose. This parameter corresponds to the following invocation of Orion:

```
%> ./orion -run advanced -testname mytest \
     -num_disks NUM_DISKS \
     -size_small 8 -size_large 1024 -type rand \
     -simulate concat -write 0 -duration 60 \
     -matrix basic
```

> **normal:** Same as `-simple`, but also generates combinations of the small random I/O and large random I/O workloads for a range of loads. The only optional parameters that can be specified at this run level are -cache_size and –verbose. This parameter corresponds to the following invocation of Orion:

```
%> ./orion -run advanced -testname mytest \
     -num_disks NUM_DISKS \
     -size_small 8 -size_large 1024 -type rand \
     -simulate concat -write 0 -duration 60 \
     -matrix detailed
```

> **advanced:** Indicates that the test parameters will be specified by the user. Any of the optional parameters can be specified at this run level.

testname: Identifier for the test. The input file containing the disk or file names must be named <testname>.lun. The output files will be named with the prefix <testname>_.

num_disks: Actual number of physical disks used by the test. Used to generate a range for the load.

OPTIONAL INPUT PARAMETERS

help: Prints Orion help information. All other options are ignored when help is specified.

size_small: Size of the I/Os (in KB) for the Small Random I/O workload. (Default is 8).

size_large: Size of the I/Os (in KB) for the Large Random or Sequential I/O workload. (Default is 1024).

type: Type of the Large I/O workload. (Default is `rand`):

> **rand:** Large Random I/O workload.
>
> **seq:** Large Sequential I/O workload.

num_streamIO: Number of outstanding I/Os per sequential stream. Only valid for `-type seq`. (Default is 1).

simulate: Data layout to simulate for Large Sequential I/O workload. [1]

> **concat:** A virtual volume is simulated by serially chaining the specified LUNs. A sequential test over this virtual volume will go from some point to the end of one LUN, followed by the beginning to end of the next LUN, etc.
>
> **raid0:** A virtual volume is simulated by striping across the specified LUNs. The stripe depth is 1M by default (to match the Oracle ASM stripe depth) and can be changed by the `-stripe` parameter.

write: Percentage of I/Os that are writes; the rest being reads. This parameter applies to both the Large and Small I/O workloads. For Large Sequential I/Os, each stream is either read-only or write-only; the parameter specifies the percentage of streams that are write-only. The data written to disk is garbage and unrelated to any existing data on the disk. **WARNING: WRITE TESTS WILL OBLITERATE ALL DATA ON THE SPECIFIED LUNS.**

cache_size: Size of the storage array's read or write cache (in MB). For Large Sequential I/O workloads, Orion will warm the cache by doing random large I/Os before each data point. It uses the cache size to determine the duration for this cache warming operation. If not specified, warming will occur for a default amount of time. If set to 0, no cache warming will be done. (Default is not specified, which means warming for a default amount of time).

duration: Duration to test each data point in seconds. (Default is 60).

matrix: Type of mixed workloads to test over a range of loads. (Default is detailed).

> **basic:** No mixed workload. The Small Random and Large Random/Sequential workloads will be tested separately.
>
> **detailed:** Small Random and Large Random/Sequential workloads will be tested in combination.
>
> **point:** A single data point with *S* outstanding Small Random I/Os and *L* outstanding Large Random I/Os or sequential streams. *S* is set by the `num_small` parameter. *L* is set by the `num_large` parameter.
>
> **col:** Large Random/Sequential workloads only.
>
> **row:** Small Random workloads only.
>
> **max:** Same as `detailed`, but only tests the workload at the maximum load, specified by the `num_small` and `num_large` parameters.

num_small: Maximum number of outstanding I/Os for the Small Random I/O workload. Can only be specified when `matrix` is `col`, `point`, or `max`.

num_large: Maximum number of outstanding I/Os for the Large Random I/O workload or number of concurrent large I/Os per stream. Can only be specified when `matrix` is `row`, `point`, or `max`.

verbose: Prints progress and status information to standard output.

---

[1] The offsets for I/Os are determined as follows:

For Small Random and Large Random workloads:

- The LUNs are concatenated into a single virtual LUN (VLUN) and random offsets are chosen within the VLUN.

For Large Sequential workloads:

- With striping (`-simulate raid0`). The LUNs are used to create a single striped VLUN. With no concurrent Small Random workload, the sequential streams start at fixed offsets within the striped VLUN. For *n* streams, stream *i* will start at offset VLUNsize * (*i* + 1) / (*n* + 1), except when *n* is 1, in which case the single stream will start at offset 0. With a concurrent Small Random workload, streams start at random offsets within the striped VLUN.

- Without striping (`-simulate CONCAT`). The LUNs are concatenated into a single VLUN. The streams start at random offsets within the single VLUN.

COMMAND-LINE EXAMPLES

- For a preliminary run to understand your storage performance with read-only, small and large random I/O workloads:

  ```
  $ orion -run simple -testname mytest -num_disks 8
  ```

- Similar to the above, but with a mixed small and large random I/O workload:

  ```
  $ orion -run normal -testname mytest -num_disks 12
  ```

- To generate combinations of 32KB and 1MB reads to random locations:

  ```
  $ orion -run advanced -testname mytest -num_disks 6  -size_small 32 \
  -size_large 1024 -type rand -matrix detailed
  ```

- To generate multiple sequential 1MB write streams, simulating 1MB RAID-0 stripes:

  ```
  $ orion -run advanced -testname mytest -num_disks 15 -simulate raid0 \
  -stripe 1024 -write 100 -type seq -matrix col -num_small 0
  ```

TROUBLE-SHOOTING

- If you are getting an I/O error on one or more of the volumes specified in the <testname>.lun file:
  - Verify that you can access it in the same mode as the test (read or write) using a file copy program such as dd.
  - Verify that your host operating system version is capable of doing asynchronous I/O.
  - On Linux and Solaris, the library libaio needs to be in one of the standard lib directories or accessible through the shell environment's library path variable (usually LD_LIBRARY_PATH or LIBPATH, depending on your shell).
- If you are running on NAS storage:
  - The file system must be properly mounted for Orion to run. Please consult your Oracle Installation Guide for directions (for example, Appendix B "Using NAS Devices" in the Database Installation Guide for Linux x86).
  - The mytest.lun file should contain one or more paths of existing files. Orion will not work on directories or mount points. The file has to be large enough for a meaningful test. The size of this file should represent the eventual expected size of your datafiles (say, after a few years of use).
  - You may see poor performance doing asynchronous I/O over NFS on Linux (including 2.6 kernels).
  - If you're doing read tests and the reads are hitting blocks of the file that were not initialized or previously written to, some smart NAS systems may "fake" the read by returning you zeroed-out blocks. The workaround is to write all blocks (using a tool like dd) before doing the read test.
- If you are running Orion on Windows:
  - Testing on raw partitions requires temporarily mapping the partitions to drive letters and specifying these drive letters (like \\.\x:)in the test.lun file.
- If you are running an Orion 32-bit Linux/x86 binary on an x86_64 machine:
  - Please copy a 32-bit libaio.so file from a 32-bit machine running the same Linux version.
- The mytestIf you are testing with a lot of disks (num_disks greater than around 30):
  - You should use the –duration option (see the optional parameters section for more details) to specify a long duration (like 120 seconds or more) for each data point. Since Orion tries to keep all spindles running at a particular load level, each data point requires a ramp-up time , which implies a longer duration for the test.

    - You may get the following error message, instructing you to increase the duration time, but we suggest doing it proactively:

      ```
      Specify a longer -duration value.
      ```

      A duration of 2x the number of spindles seems to be a good rule of thumb. Depending on your disk technology, your platform may need more or less time.

- If you get an error about libraries being used by Orion:
    - Linux/Solaris: See I/O error troubleshooting above.
    - **NT-Only:** Do not move/remove the Oracle libraries included in the distribution. These need to be in the same directory as orion.exe.
- If you are seeing performance numbers that are "unbelievably good":
    - You may have a large read and/or write cache somewhere between the Orion program and the disk spindles. Typically, the storage array controller has the biggest effect. Find out the size of this cache and use the –cache_size advanced option to specify it to Orion (see the optional parameters section for more details).
    - The total size of your volumes may be really small compared to one or more caches along the way. Try to turn off the cache. This is needed if the other volumes sharing your storage will see significant I/O activity in a production environment (and end up using large parts of the shared cache).
- If Orion is reporting a long estimated run time:
    - The run time increases when –num_disks is high. Orion internally uses a linear formula to determine how long it will take to saturate the given number of disks.
    - The –cache_size parameter affects the run time, even when it's not specified. Orion does cache warming for two minutes per data point by default. If you have turned off your cache (which we recommend for read caches where possible), specify –cache_size 0.
    - The run time increases when a long –duration value is specified, as expected.

Appendix A: Characterizing The Database I/O Load[2]

To properly configure your database storage, you must understand your database's performance requirements. In particular, you must answer the following questions:

1. **Will the I/O requests be primarily single-block or multi-block?**

   Databases issue multi-block I/Os when performing the following types of operations: parallel queries, queries on large tables that require table scans, direct data loads, backups, and restores. In general, DSS and data warehouse environments issue large amounts of multi-block I/Os whereas OLTP databases primarily issue single-block I/Os.
2. **What is your average and peak I/Os per second (IOPS) requirement? What percentage of this traffic are writes?**
3. **What is your average and peak throughput (in MBPS) requirement? What percentage of this traffic are writes?**

If your database's I/O requests are primarily single-block, then you should focus on ensuring that the storage can accommodate your I/O request rate (IOPS). If they are primarily multi-block, then you should focus on the storage's throughput capacity (MBPS).

For an existing 10gR2 database, you can characterize your I/O traffic by looking at the database statistics in the V$SYSSTAT view, as referenced by the names given below. These statistics are cumulative values that should be sampled during both the typical and peak periods.

- single-block reads: "physical read total IO requests" – "physical read total multi block requests"
- multi-block reads: "physical read total multi block requests"
- bytes read: "physical read total bytes"
- single-block writes: "physical write total IO requests" – "physical write total multi block requests"
- multi-block writes: "physical write total multi block requests"
- bytes written: "physical write total bytes"

For an existing pre-10gR2 database, the I/O statistics are specified in multiple views. The bulk of the I/O traffic is described as follows:

- single-block data file reads: V$FILESTAT.SINGLEBLKRDS specifies the number of such I/O requests.
- multi-block data file reads: V$FILESTAT.PHYRDS – V$FILESTAT.SINGLEBLKRDS specifies the number of such I/O requests.
- single-block data file writes: V$FILESTAT.PHYWRTS specifies the number of such I/O requests.
- multi-block data file writes: V$FILESTAT.PHYBLKWRT specifies the number of DBWR writes plus the number of direct I/O blocks written. Unfortunately, there isn't a way to derive the number of multi-block I/O requests, but in general, direct I/Os are multi-block I/Os. The number of direct I/O blocks written is V$FILESTAT.PHYBLKWRT - V$FILESTAT.PHYWRTS.
- redo log writes: in V$SYSSTAT, "redo blocks written" specifies the number of blocks written and "redo writes" specifies the number of I/O requests.
- backup I/Os: in V$BACKUP_ASYNC_IO and V$BACKUP_SYNC_IO, the IO_COUNT field specifies the number of I/O requests and the TOTAL_BYTES field specifies the number of bytes read or written. Note that each row of this view corresponds to a data file, the aggregate over all data files, or the output backup piece.

[2] This section is taken from the Oracle white paper "Best Practices for a Low-Cost Storage Grid for Oracle Databases". We recommend that you reference this paper for the latest version of this information.

- flashback log I/Os: In V$FLASHBACK_DATABASE_STAT, FLASHBACK_DATA, DB_DATA, and REDO_DATA show the number of bytes read or written from the flashback logs, data files, and redo logs, respectively, in the given time interval. In V$SYSSTAT, the "flashback log writes" statistic specifies the number of write I/O requests to the flashback log.

Using this data, you can estimate the read and write IOPS requirement from the number of physical reads and writes issued in a given duration of time, both at peak and normal loads. You can estimate the read and write MBPS requirement from the number of bytes read and written per second, again at both peak and normal loads. The number of writes compared to the number of reads indicates the "write percentage" to be specified to Orion. For example, if your applications typically do 600 writes and 200 reads in a given duration, your write percentage would be 75% (-write 75). ***Please read the warnings in the –write option description before doing write tests.***

Note that these statistics do not include all I/O traffic, such as control file and archiver-generated I/Os.

APPENDIX B: NOTES FOR DATA WAREHOUSING

I/O performance should always be a key consideration for data warehouse designers and administrators. The typical workload in a data warehouse is especially I/O intensive, with operations such as queries over large volumes of data, large data loads and index builds and creation of materialized views. The underlying I/O system for a data warehouse should be designed to meet these heavy requirements.

Storage configurations for a data warehouse should be chosen based on the I/O bandwidth that they can provide, and not necessarily on their overall storage capacity. The capacity of individual disk drives is growing faster than the I/O throughput rates provided by those disks, leading to a situation in which a small number of disks can store a large volume of data, but cannot provide the same I/O throughput as a larger number of small disks. You get maximal I/O bandwidth by having multiple disks and channels contribute to the heavy database operations. Striping data files across devices (ideally, all devices) is a way to achieve this. Implement a large stripe size (1 MB) in order to ensure that the time to position the disk is a small percentage of the time to transfer the data.

The workload for a data warehouse is typically characterized by sequential I/O throughput, issued by multiple processes. You can simulate this type of workload with Orion. Depending on the type of system you plan to build, you should run multiple different I/O simulations. For example,

- Daily workload when end-users and/or other applications query the system: read-only workload with possibly many individual parallel I/Os.
- The data load, when end-users may or may not access the system: write workload with possibly parallel reads (by the load programs and/or by end-users).
- Index and materialized view builds, when end-users may or may not access the system: read/write workload.
- Backups: read workload with likely few other processes, but a possibly high degree of parallelism.

Use the following options in the advanced user mode of Orion to simulate the different data warehouse-like workload types:

- **run:** use 'advanced' in order to simulate only large sequential I/Os.
- **large:** the I/O size in KB for large sequential loads. Set this parameter to a multiple of the operating system I/O size. For a data warehouse, you should plan for single I/O requests that are as large as possible. On most platforms, this size is 1 MB.
- **type:** use 'seq' to simulate a large sequential I/O workload.
- **num_streamIO:** increase this parameter in order to simulate parallel execution for individual operations. Specify a degree of parallelism that you plan to use for your database operations. A good starting point for the degree of parallelism is the number of CPUs on your system multiplied by the number of parallel threads per CPU.

- **simulate:** use 'CONCAT' if the devices are already striped as they are presented to the database, e.g. if striping takes place on the hardware level or through a volume manager. Use 'raid0' for devices that have yet to be striped, for example by Oracle Automatic Storage Manager. The stripe size for 'raid0' defaults to 1 MB.
- **write:** specify the percentage of I/Os that you want to be writes. Take your data load programs into account and the access to the system during the load window in order to set the percentage. WARNING: WRITE TESTS WILL OBLITERATE ALL DATA ON THE LUNS.
- **matrix:** use 'point' to simulate an individual sequential workload, or use 'col' to simulate an increasing number of large sequential workloads.
- **num_large:** specify the maximum number of large I/Os.

In a clustered environment, you will have to invoke Orion in parallel on all nodes in order to simulate a clustered workload. For example, the following example is an Orion run for a typical data warehouse workload. To get started, a file orion14.lun was created, with the following contents:

```
/dev/vx/rdsk/asm_vol1_1500m
/dev/vx/rdsk/asm_vol2_1500m
/dev/vx/rdsk/asm_vol3_1500m
/dev/vx/rdsk/asm_vol4_1500m
```

These 4 disks are individual volumes that are not striped on the hardware level, nor on a volume manager level. ASM would be used to stripe data files across these 4 volumes. The following command invokes Orion:

```
./orion -run advanced \
-testname orion14 \
-matrix point \
-num_small 0 \
-num_large 4 \
-size_large 1024 \
-num_disks 4 \
-type seq \
-num_streamIO 8 \
-simulate raid0 \
-cache_size 0 \
-verbose
```

The run simulates 4 parallel sessions (-num_large 4) running a statement with a degree of parallelism of 8 (-num_streamIO 8). Orion simulates a raid0 striping. The internal disks in this case do not have cache. As a result of running this command, the file orion14_summary.txt is generated with the following contents:

```
ORION VERSION 10.2

Command line:
-run advanced -testname orion14 -matrix point -num_large 4 -size_large 1024 -
num_disks 4 -type seq -num_streamIO 8 -simulate raid0 -cache_size 0 -verbose

This maps to this test:
Test: orion14
Small IO size: 8 KB
Large IO size: 1024 KB
IO Types: Small Random IOs, Large Sequential Streams
Number of Concurrent IOs Per Stream: 8
Force streams to separate disks: No
Simulated Array Type: RAID 0
```

```
Stripe Depth: 1024 KB
Write: 0%
Cache Size: 0 MB
Duration for each Data Point: 60 seconds
Small Columns:, 0
Large Columns:, 4
Total Data Points: 1

Name: /dev/vx/rdsk/asm_vol1_1500m       Size: 1572864000
Name: /dev/vx/rdsk/asm_vol2_1500m       Size: 1573912576
Name: /dev/vx/rdsk/asm_vol3_1500m       Size: 1573912576
Name: /dev/vx/rdsk/asm_vol4_1500m       Size: 1573912576
4 FILEs found.

Maximum Large MBPS=57.30 @ Small=0 and Large=4
```

In other words, the maximum throughput for this specific case with that workload is 57.30 MBps. In ideal conditions, Oracle will be able to achieve up to about 95% of that number. For this particular case, having 4 parallel sessions running the following statement would approach the same throughput:

```
select /*+ NO_MERGE(sales) */ count(*) from
   (select /*+ FULL(s) PARALLEL (s,8) */ * from all_sales s) sales
```

Note that the Oracle database has several optimizations that would make a statement like this run faster in case it wasn't forced to read the entire table.

In a well-balanced data warehouse hardware configuration, there is sufficient I/O bandwidth to feed the CPUs. As a starting point, you can use the basic rule that every GHz of CPU power can drive at least 100 MBps. I.e., for a single server configuration with four 3 GHz CPUs, your storage configuration should at least be able to provide 4 * 3 * 100 = 1200 MBps throughput. This number should be multiplied by the number of nodes in a RAC configuration.