written, can represent your workload. If your applications perform well on the new platform, then there is a higher probability of the platform's acceptance and ultimate success. The worst possible outcome for the consolidated platform would be the lack of acceptance!

■ **Caution** Some of the benchmarks listed and described below can perform destructive testing and/or cause high system load. Always ensure that your benchmark does not have any negative impact on your environment! Do not benchmark outside your controlled and isolated lab environment! Always carefully read the documentation that comes with the benchmark. And be especially careful when it comes to writing benchmarks

---

If you cannot find representative workloads from within the range of applications in use in your organization, you may have to resort to an off-the-shelf benchmark. Large Enterprise Resource Planning systems usually come with a benchmark, but those systems are unlikely candidates for your consolidation platform. If you want to test different components of your system, specialized benchmarks come to mind, but those don't test the platform end-to-end. For storage related testing you could use these benchmarks, for example:

- iozone
- bonnie++
- hdbench
- fio
- And countless others more . . .

Of all the available storage benchmarks, FIO sounds very interesting. It is a very flexible benchmark written by Jens Axboe to test different I/O schedulers in Linux and will be presented in more detail below.

Network-related benchmarks, for example, include the officially Oracle-sanctioned `iperf` and others. Kevin Closson's Silly Little Benchmark tests memory performance, and so does the University of Virginia's STREAM benchmark. The chips on the mainboard, including the CPU and also cooling, can be tested using the High Performance Linpack benchmark, which is often used in High Performance Computing (HPC).

Each of these benchmarks can be used to get the bigger picture, but none is really suited to assess the system's qualities when used in isolation. More information about the suitability of the platform, especially in respect to storage performance, can be obtained by using Oracle IO Numbers (ORION) or the Silly Little Oracle Benchmark (SLOB) written by Kevin Closson. The below sections are a small selection of available benchmarks.

FIO

FIO is an intelligent I/O testing platform written by Jens Axboe, whom we also have to thank for the Linux I/O schedulers and much more. Out of all the I/O benchmarks, I like FIO because it is very flexible and offers a wealth of output for the performance analyst. As with most of I/O related benchmarks, FIO works best in conjunction with other tools to get a more complete picture.

What is great about the tool is the flexibility, but it requires a little more time to learn all the ins and outs of it. However, if you take the time to learn FIO, you will automatically learn more about Linux as well. FIO benchmark runs are controlled by plain text files with instructions, called a job file. A sample file is shown here; it will be used later in the chapter:

```
[oracle@server1 fio-2.1]# cat rand-read-sync.fio
[random-read-sync]
rw=randread
```

```
size=1024m
bs=8k
directory=/u01/fio/data
ioengine=sync
iodepth=8
direct=1
invalidate=1
ioscheduler=noop
```

In plain English, the job named “random-read-sync” will use the random-read workload, creates a file of 1 GB in size in the directory `/u01/fio/data/`, and uses a block size of 8 kb, which matches Oracle’s standard database block size. A maximum of 8 outstanding I/O requests are allowed, and the Linux I/O scheduler to be used is the `noop` scheduler since the storage in this case is non-rotational. The use of direct I/O is also requested, and the page cache is invalidated first to avoid file system buffer hits for consistency with the other scripts in the test harness—direct I/O will bypass with buffer cache anyway, so strictly speaking, the directive is redundant.

Linux can use different ways to submit I/O to the storage subsystem—synchronous and asynchronous. Synchronous I/O is also referred to as blocking I/O, since the caller has to wait for the request to finish. Oracle will always use synchronous I/O for single block reads such as index lookups. This is true, for example, even when asynchronous I/O is enabled. The overhead of setting up an asynchronous request is probably not worth it for a single block read.

There are situations wherein synchronous processing of I/O requests does not scale well enough. System designers therefore introduced asynchronous I/O. The name says it all: when you are asynchronously issuing I/O requests, the requestor can continue with other tasks and will “reap” the outstanding I/O request later. This approach greatly enhances concurrency but, at the same time, makes tracing a little more difficult. As you will also see, the latency of individual I/O requests increases with a larger queue depth. Asynchronous I/O is available on Oracle Linux with the libaio package.

Another I/O variant is called direct I/O, which instructs a process to bypass the file system cache. The classic UNIX file system buffer cache is known as the page cache in Linux. This area of memory is the reason why users do not see free memory in Linux. Those portions of memory that are not needed by applications will be used for the page cache instead.

Unless instructed otherwise, the kernel will cache information read from disk in said page cache. This cache is not dedicated to a single process allowing other processes to benefit from it as well. The page cache is a great concept for the many Linux applications but not necessarily beneficial for the Oracle database. Why not? Remember from the preceding introduction that the result of regular file I/O is stored in the page cache in addition to being copied to the user process’s buffers. This is wasteful in the context of Oracle since Oracle already employs its own buffer cache for data read from disk. The good intention of a file-system page cache is not needed for the Oracle database since it already has its own cache to counter the relatively slow disk I/O operations.

There are cases in which enabling direct I/O causes performance problems, but those can easily be trapped in test and development environments. However, using direct I/O allows the performance analyst to get more accurate information from performance pages in Oracle. If you see a 1 ms response time from the storage array, you cannot be sure if that is a great response time from the array or rather a cached block from the operating system. If possible, you should consider enabling direct I/O after ensuring that it does not cause performance problems.

To demonstrate the difference between synchronous and asynchronous I/O, consider the following job file for asynchronous I/O.

```
[oracle@server1 fio-2.1]$ cat rand-read-async.fio
[random-read-async]
rw=randread
size=1024m
bs=8k
directory=/u01/fio/data
```

```
ioengine=libaio
iodepth=8
direct=1
invalidate=1
ioscheduler=noop
```

Let's compare the two—some information has been removed in an effort not to clutter the output. First, the synchronous benchmark:

```
[oracle@server1 fio-2.1]$ ./fio rand-read-sync.fio
random-read-sync: (g=0): rw=randread, bs=8K-8K/8K-8K/8K-8K, ioengine=sync, iodepth=8
fio-2.1
Starting 1 process
Jobs: 1 (f=1): [r] [100.0% done] [39240KB/0KB/0KB /s] [4905/0/0 iops] [eta 00m:00s]
random-read-sync: (groupid=0, jobs=1): err= 0: pid=4206: Mon May 27 23:03:22 2013
  read : io=1024.0MB, bw=39012KB/s, iops=4876, runt= 26878msec
    clat (usec): min=71, max=218779, avg=203.77, stdev=605.20
     lat (usec): min=71, max=218780, avg=203.85, stdev=605.20
    clat percentiles (usec):
     |  1.00th=[  189],  5.00th=[  191], 10.00th=[  191], 20.00th=[  193],
     | 30.00th=[  195], 40.00th=[  197], 50.00th=[  205], 60.00th=[  207],
     | 70.00th=[  209], 80.00th=[  211], 90.00th=[  213], 95.00th=[  215],
     | 99.00th=[  225], 99.50th=[  233], 99.90th=[  294], 99.95th=[  318],
     | 99.99th=[  382]
    bw (KB  /s): min=20944, max=39520, per=100.00%, avg=39034.87, stdev=2538.87
    lat (usec) : 100=0.08%, 250=99.63%, 500=0.29%, 750=0.01%, 1000=0.01%
    lat (msec) : 4=0.01%, 20=0.01%, 250=0.01%
  cpu          : usr=0.71%, sys=6.17%, ctx=131107, majf=0, minf=27
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=131072/w=0/d=0, short=r=0/w=0/d=0

Run status group 0 (all jobs):
   READ: io=1024.0MB, aggrb=39012KB/s, minb=39012KB/s, maxb=39012KB/s,
         mint=26878msec, maxt=26878msec

Disk stats (read/write):
  sda: ios=130050/20, merge=0/4, ticks=25188/12, in_queue=25175, util=94.12%
```

The important information here is that the test itself took 26,878 milliseconds to complete, and the storage device completed an average of 4,876 I/O operations per second for an average bandwidth of 39,012KB/s. Latencies are broken down into scheduling latency (not applicable for synchronous I/O— you will see it in the below output) and completion latency. The last number, recorded as "lat" in the output above, is the complete latency and should be the sum of scheduling plus completion latency.

Other important information is that out CPU has not been terribly busy when it executed the benchmark, but this is an average over all cores. The IO depths row shows the IO depth over the duration of the benchmark execution. As you can see, the use of synchronous I/O mandates an I/O depth of 1. The "issued" row lists how many reads and writes have been issued. Finally, the result is repeated again at the bottom of the output. Let's compare this with the asynchronous test:

```
[oracle@server1 fio-2.1]$ ./fio rand-read-async.fio
random-read-async: (g=0): rw=randread, bs=8K-8K/8K-8K/8K-8K, ioengine=libaio, iodepth=8
fio-2.1
Starting 1 process
random-read-async: Laying out IO file(s) (1 file(s) / 1024MB)
Jobs: 1 (f=1): [r] [100.0% done] [151.5MB/0KB/0KB /s] [19.4K/0/0 iops] [eta 00m:00s]
random-read-async: (groupid=0, jobs=1): err= 0: pid=4211: Mon May 27 23:04:28 2013
  read : io=1024.0MB, bw=149030KB/s, iops=18628, runt=  7036msec
    slat (usec): min=5, max=222754, avg= 7.89, stdev=616.46
    clat (usec): min=198, max=14192, avg=408.59, stdev=114.53
     lat (usec): min=228, max=223110, avg=416.61, stdev=626.83
    clat percentiles (usec):
     |  1.00th=[  278],  5.00th=[  306], 10.00th=[  322], 20.00th=[  362],
     | 30.00th=[  398], 40.00th=[  410], 50.00th=[  414], 60.00th=[  422],
     | 70.00th=[  430], 80.00th=[  442], 90.00th=[  470], 95.00th=[  490],
     | 99.00th=[  548], 99.50th=[  580], 99.90th=[  692], 99.95th=[  788],
     | 99.99th=[ 1064]
    bw (KB  /s): min=81328, max=155328, per=100.00%, avg=149248.00, stdev=19559.97
    lat (usec) : 250=0.01%, 500=96.31%, 750=3.61%, 1000=0.06%
    lat (msec) : 2=0.01%, 20=0.01%
  cpu          : usr=3.80%, sys=21.68%, ctx=129656, majf=0, minf=40
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=100.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.1%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=131072/w=0/d=0, short=r=0/w=0/d=0

Run status group 0 (all jobs):
   READ: io=1024.0MB, aggrb=149030KB/s, minb=149030KB/s, maxb=149030KB/s,
         mint=7036msec, maxt=7036msec

Disk stats (read/write):
  sda: ios=126931/0, merge=0/0, ticks=50572/0, in_queue=50547, util=95.22%
```

Apart from the fact that the test completed a lot faster—7,036msec vs. 26,878msec—you can see that bandwidth is a lot higher. The libaio benchmark provides a lot more IOPS as well: 18,628 vs. 4,876. A doubling of the IO depth yields better bandwidth with the asynchronous case at the expense of the latency of individual requests. You do not need to spend too much time trying to find the maximum I/O depth on your platform, since Oracle will transparently make use of the I/O subsystem, and you should probably not change any potential underscore parameter.

You should also bear in mind that maximizing I/O operations per second is not the only target variable when optimizing storage. A queue depth of 32, for example, will provide a large number of IOPS, but this number means little without the corresponding response time. Compare the output below, which has been generated with the same job file as before but a queue depth of 32 instead of 8:

```
[oracle@server1 fio-2.1]$ ./fio rand-read-async-32.fio
random-read-async: (g=0): rw=randread, bs=8K-8K/8K-8K/8K-8K, ioengine=libaio, iodepth=32
[...]
  read : io=1024.0MB, bw=196768KB/s, iops=24595, runt=  5329msec
```

```
[...]
    lat (usec) : 500=0.01%, 750=0.01%, 1000=0.01%
    lat (msec) : 2=99.84%, 4=0.13%
```

The number of IOPS has increased from 18,628 to 24,595, and the execution time is down to 5,329 milliseconds instead of 7,036, but this has come at a cost. Instead of microseconds, 99.84% of the I/O requests now complete in milliseconds.

When you are working with storage and Linux, FIO is a great tool to help you understand the storage solution as well as the hardware connected against it. FIO has a lot more to offer with regards to workloads. You can read and write sequentially or randomly, and you can mix these as well—even within a single benchmark run! The permutations of I/O type, block sizes, direct, and buffered I/O, combined with the options to run multiple jobs and use differently sized memory pages, etc., make FIO one of the best benchmark tools available. There is a great README as part of FIO, and its author has a HOWTO document, as well, for those who want to know all the detail: `http://www.bluestop.org/fio/HOWTO.txt`.

Oracle I/O numbers

The ORION tool has been available for quite some time, initially as a separate download from Oracle's website and now as part of the database installation. This, in a way, is a shame since a lot of its attractiveness initially came from the fact that a host did *not* require a database installation. If you can live with a different, lower, version number, then you can still download ORION for almost all platforms from Oracle's OTN website. At the time of this writing, the below URL allowed you to download the binaries: `http://www.oracle.com/technetwork/topics/index-089595.html`. The above link also allows you to download the ORION user's guide, which is more comprehensive than this section. For 11.2 onwards you find the ORION documentation as part of the official documentation set in the "Performance Tuning Guide". Instead of the 11.1 binaries downloadable as a standalone package, this book will use the ones provided with the Oracle server installation.

ORION works best during the evaluation phase of a new platform, without user-data on your logical unit numbers (LUNs). (As with all I/O benchmarking tools, write testing is **destructive**!) The package will use asynchronous I/O and large pages, where possible, for reads and writes to concurrently submit I/O requests to the operating system. Asynchronous I/O in this context means the use of `libaio` on Linux or equivalent library as discussed in the FIO benchmark section above. You can quite easily see that the ORION slave processes use `io_submit` and `io_getevents` by executing `strace` on them:

```
[root@server1 ~]# strace -cp 28168
Process 28168 attached - interrupt to quit
^CProcess 28168 detached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 97.79    0.323254           4     86925           io_submit
  2.11    0.006968           1     12586           io_getevents
  0.10    0.000345           0     25172           times
------ ----------- ----------- --------- --------- ----------------
100.00    0.330567                124683           total
[root@server1 ~]#
```

The process with ID 28168, to which the strace session attached, was a running ORION session.