Oracle and storage IOs, explanations and experience at CERN

Eric Grancher

CERN Geneva 23 CH-1211 Switzerland

Eric.Grancher@cern.ch

Abstract. The Oracle database system is used extensively in the High Energy Physics community. Critical to the efficient running of these databases is the storage subsystem, and over the years Oracle has introduced new ways to access and manage this storage, e.g. ASM (Oracle database version 10.1), Direct NFS (Oracle database version 11.1), and Exadata (Oracle database version 11.1). This paper presents CERN's experience over the past few years with the different storage access and management features, and gives a comparison of each functionality. Also compared are the different solutions used at CERN, and the Tier 1 sites for storing Oracle databases.

1. Oracle in Accelerator, High Energy Physics and Worldwide LHC Computing Grid

The Oracle [1] database has been used since 1982 at CERN and in other High Energy Physics laboratories. It has been chosen following an evaluation essentially for its ability to be part of the accelerator controls systems, see [2] for a scanned version of the original document.

Since then, the Oracle database software has been selected as a key element of the accelerator controls systems, for the CERN laboratory infrastructure (examples of it are the CERN Engineering and Equipment Data Management Service [3] and the administrative applications) as well as for the Large Hadron Collider (LHC) experiments. The LHC experiments have based a significant part of the online and offline systems on Oracle, as described in [4], making use for part of it of the Streams functionality ([5]). It has been recently demonstrated in [6] that the Oracle database system can be used as a storage and analysis foundation for large and intensive accelerators and experiments control systems.

The Worldwide LHC computing grid (WLCG) consists of three main layers or "tiers", as described in [7]. The Oracle database system is deployed in ten of the eleven Tier-1 sites.

2. Oracle and the IO subsystem

By nature, a database system is dependent on the Input Output storage system. The Input Output (IO) storage system is often called "disk subsystem", with the flash Solid State Drive (SSD) devices coming increasingly as replacement for the rotating drives, the IO subsystem is a more adequate term.

The Oracle database system makes use of the different types of IO subsystems available, presented to the host, or hosts in case of clustering. In the past years, the operating system vendors or developers have added features to their system that have enabled the Oracle database to make better use of the computing resources and provide faster response time. This is for example the case of Asynchronous and direct IO features. A description of the major types of IO subsystems as used by the Oracle database is given is the next section.

2.1. Asynchronous and direct IO

The two options are now widely available on all platforms where Oracle is supported, it is either used by default, for example with Automated Storage Management (ASM), or requires to be set explicitly (the `setall` value for the database parameter `FILESYSTEMIO_OPTIONS` will enable both asynchronous and direct IO).

Asynchronous IO enables the database software to launch several IO operations and then wait for their completion; it is beneficial for writing to the redo logs and the database files, see [8] for further explanation. The implementation depends on the operating system, some operating systems use the Portable Operating System Interface [for UNIX] (POSIX) libraries, Linux uses the `libaio` library. In [9], there are examples on how to identify which system calls are used.

The direct IO feature is available on some platforms and enables to bypass the system buffer cache, it almost always provides a performance gain and reduces the CPU usage, the buffer cache already provides the best approach for data caching, see [10] for further explanation. The exception that we have observed is for the Large Object Binary (LOB) entities which are not cached by default by the Oracle database system and for which the filesystem cache can provide a significant performance improvement. If faced with such a case, the LOB columns can be tagged to be cached in the buffer cache. One has to be careful with such a change that the activity of loading lots of data in the buffer cache does not lead to poor performance for other objects. Direct IO is enabled when the file is opened, using the `O_DIRECT` tag of the `open(2)` system call.

2.2. Real Application Cluster (RAC) and IO subsystem

With Real Application Cluster, one of the pre-requisite and fundamental requirements is a symmetric access to the datafiles and online redo logs. It means that all nodes in the cluster should be able to access the storage in a symmetrical way. It implies that the usage of Directly Attached Storage (typically disks behind a set of controller cards) is not possible. In the early implementations of RAC (or Oracle Parallel Server, the predecessor of RAC) this requirements has often been a source of increased complexity and human errors. An example of issues linked with this requirement is presented in section 2.3.3.

One can classify the type of IO subsystems in two main classes: block device access and file based access.

2.3. Block devices

Block device access, implemented as Fiber Channel Storage Area Network, Internet Small Computer System Interface (iSCSI) or Fibre Channel over Ethernet (FCoE) is a very common access deployment method for Oracle databases. The underlying media can be Fibre Channel, Ethernet or Infiniband.

2.3.1. Raw devices

Raw device access was initially widely recommended for high performance Oracle implementations on UNIX for its minimal overhead regarding IO subsystem access. For Oracle 9i Real Application Cluster, it was the only option if neither a cluster filesystem nor a Network File System (NFS) capable storage subsystem was used.

The implementation is different depending on the operating system: Solaris has for example a `/dev/rdsk/cAtBdCsD` device for each `/dev/dsk/cAtBdCsD`, Linux has the `raw(8)` command to create raw device bindings `/dev/raw/raw<N>`.

Oracle has announced in [11] that it will not support anymore using raw devices for creating new databases with the database configuration assistant as of the 11.2 release.

2.3.2. ASM management

Automatic Storage Management is a feature of the Oracle database software which has been released as of the Oracle database version 10.1. It consists in a volume management system implemented as specialised Oracle instance which provides access to the database instance. For “normal redundancy” disk groups, it implements the "Stripe And Mirror Everything" paradigm as well as direct and asynchronous IO. It has been a major step forward for the simplification of Oracle cluster deployment and datafile management.

CERN had tested early versions of ASM and has deployed ASM on Serial ATA (SATA) disks based, Fiber Channel connected disk arrays as described in [12]. All of the WLCG Tier 1 sites deploying the Oracle database system are using ASM for part or all of the databases. The Port d'Informació Científica (PIC) is using ASM on top of NAS based storage systems.

2.3.3. Third party volume manager
In the early (1990s) years of Oracle UNIX deployment, both for performance and flexibility, a third party volume manager was used to create "volumes" (Veritas Volume Manager term) on top of disks, it was a key enabler to have stripping (for performance), and mirroring (for reliability).

For the early (1990s) Oracle cluster deployment, a very common approach was to use a cluster volume manager like the Veritas Cluster Volume Manager (CVM), a clustered version of the widely used Veritas Volume Manager (VxVM).

At CERN, the added complexity of a large cluster volume manager complexity has been the root cause for several incidents leading to downtime. One of the cons of such a solution is the impossibility to use the datafile auto-extensibility feature.

2.4. File bases access
File based access is the most natural deployment model for Oracle databases, with Oracle naming itself the different type of files as "datafiles", "tempfiles", "redo log files", "spfiles"... For single instance systems, it is a widely used solution, for clustered Oracle database, cluster filesystems and NFS based solutions are also widely used. Usage of file based systems also enables a central storage of log and configuration files.

2.4.1. Local filesystem
The simplest deployment method is to use the local filesystem, on Linux, at the time of the writing for this paper, ext3 is the most widely used filesystem. As of RedHat Enterprise Linux 4, deployment of Oracle databases on top of ext3 can make use of both direct and asynchronous IO.

2.4.2. Cluster Filesystem
Deployment on top of cluster filesystem provides a lot of facilities which was especially desirable for clusters before the availability of ASM in Oracle database version 10.1, indeed the typical other option to be used was to use raw devices.

Working with a cluster filesystem has been an option for OpenVMS. For UNIX/Linux this possibility has been offered as of the availability of Tru64 Cluster FileSystem. Since then, other cluster filesystems have been made available: OCFS / OCFS2 cluster filesystems from Oracle, GPFS from IBM, Veritas Cluster File System, Sun QFS, RedHat GFS, PolyServe Matrix Server. See the certification matrix is available at [13]. Academia Sinica Grid Computing (ASGC) is using OCFS2 for some of its databases.

2.4.3. Host NFS stack
NFS storage systems are providing a valuable solution, easy to use and manage for database storage. In the past years, the operating system vendors and Linux kernel developers have added a number of features like asynchronous and direct IO which leads to excellent performance. At the same time, the ubiquitous presence of 1 Gbit Ethernet per second, link aggregation, and increasingly 10 Gbit Ethernet per second leads to high bandwidth from the database instance servers to the storage systems. CERN has started using NFS based appliances for Oracle databases since 2006, see [14] for explanations about the setup and experience.

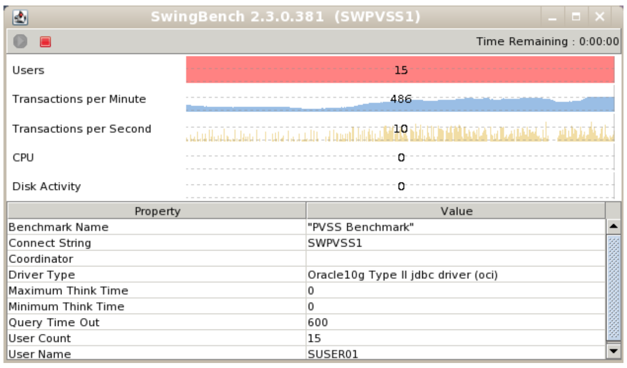
2.4.4. *Direct NFS*

As of Oracle database version 11.1, Oracle has made available the “Oracle Direct NFS client” which is an embedded (in the form of an ODM library) client in the database instance. It performs the link aggregation and fail-over of the links if unavailable. This has the advantage of a high-performance implementation with some functionality which is not available in host based NFS stack (like combined link aggregation and fail-over).

2.5. Oracle Exadata storage server

The Oracle Exadata storage server version 1 is a solution developed by Oracle and HP in which part of the processing, normally performed by the database instance, is performed at the storage system level. This includes, for example in Oracle database version 11.1, the offloading of datafile creation, column project, row selection.

CERN has tested the “HP Oracle Database Machine” which consists of a number of Exadata storage servers with database instance servers, a number of performance tests have been performed. In order push the system to its limits in a remote environment, we have developed a Swingbench ([15]) plug-in (figure 1) that provides PVSS (the Supervisory Control And Data Acquisition -SCADA- software used at CERN) workload.



SwingBench 2.3.0.381 (SWPVSS1)

Time Remaining : 0:00:00

Users	15
Transactions per Minute	486
Transactions per Second	10
CPU	0
Disk Activity	0

Property	Value
Benchmark Name	"PVSS Benchmark"
Connect String	SWPVSS1
Coordinator	
Driver Type	Oracle10g Type II jdbc driver (oci)
Maximum Think Time	0
Minimum Think Time	0
Query Time Out	600
User Count	15
User Name	SUSER01

Figure 1. Swingbench simulation of the PVSS workload.

Using this Swingbench plug-in, we have been able to stress test the Exadata system in different configurations (figure 2). For example, we have been testing the workload with the file creation offloading capabilities enabled, which is the default, or disabled.

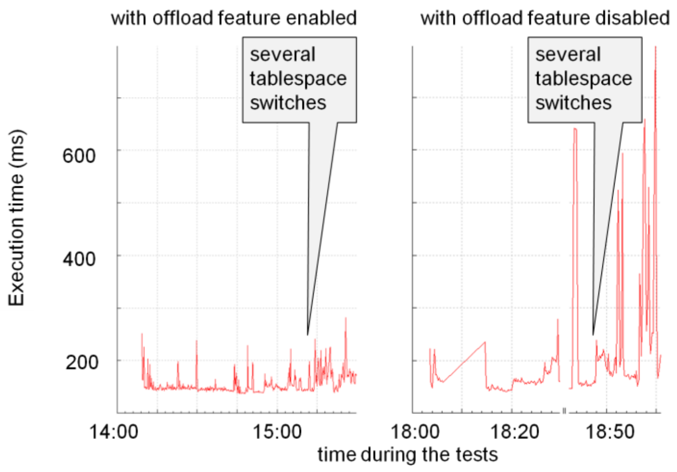


Figure 2. Comparison of the execution time of insertion (in ms) with the offload feature enabled and disabled.

In order to sustain the load, the execution time has to be less than 1000ms: indeed the system works by inserting a certain number of entries every second, if it takes more than one second to insert these entries, the system cannot sustain the load.

We can observe that, with the offload feature enabled (which is the default), one can sustain a high load with the tablespace switches (new files for data storage creation) without perturbation. Without the feature, the tablespace switches create an additional load on the server to storage connection which leads to having peaks of execution times and the overall system not being able to sustain the load in a stable way.

3. Identifying Oracle IOs

3.1. Oracle instrumentation

The Oracle database software is instrumented, it implies that, when the server process associated with a session is required to wait -on a transaction lock, a disk IO...-, it will publish this information at various places. The place to find this information can be session based aggregate values, system based aggregate values, in memory or disk sampled history (Active Session History, see next section) or it can be a traced to a file if the session has been set in a special mode.

```
Set the 10046 event at the session level or call
DBMS_MONITOR.SESSION_TRACE_ENABLE(sid,serial#,
wait, binds);

WAIT #5: nam='db file sequential read'
ela=6784 file#=6 block#=467667 blocks=1 obj#=73442
tim=1490530491532
```

Figure 3. In this example, we show that how a trace file looks like when a session is waiting for the answer of an IO operation (elapsed time in bold).

In [16], Carry Millsap describes how developers can make use of the extended SQL trace and instrumentation information to have feedback on the code written and how it executes in the database instance.

3.2. Active Session History

Active Session History is a feature of the Oracle database introduced in version 10.1. It is based on a one second sampling of the active sessions (not waiting for input from the client) where information about the SQL being executed and about the session state (on CPU or waiting for a given event) is stored. This information is stored in the memory of the instance and persisted to disk (with another sampling). A graphical description is available in [17].

As not all information is is available due to the sampling, some important information might be missing when investigating performance issues. But, for the IO patterns, we have measured and observed the fact that the Active Session History is actually providing very accurate information.

One can compare the complete instrumentation information, as retrieved from the trace file with the sampled information. We have been studying the differences on a number of systems and have observed that the collected information is very similar. In the following example (figure 4), a simple comparison is made -on the accessed segments- between Active Session History and the trace file information.

```
SQL> create table t as select p1,p2 from
v$active_session_history h where h.module like 'DATA_LOAD%'
and h.action like 'COLLECT_DN%' and h.event ='db file
```

```
sequential read' and h.sample_time>sysdate-4/24;
> grep "db file sequential read" accmeas2_j004_32116.trc  |
head -2 | awk  '{print $9"="$10}' | awk -F= '{print $2","$4}'
13,200041   |
6,587915    | data loaded in table t
SQL> select distinct e.owner, e.segment_name,
e.PARTITION_NAME,(e.bytes/1024/1024) size_MB from t,
dba_extents e where e.file_id=t.p1 and t.p2 between e.block_id
and e.block_id+e.blocks order by
e.owner,e.segment_name,e.PARTITION_NAME;
```

Figure 4. In this example, we show how one can compare the information about the objects being accessed, using Active Session History with the one second sampling with the same information about which objects are being accessed based on the 10046 trace file.

3.3. IO at the Operating System level

In order to understand how the Oracle database is performing IO operations, it is important to observe which functions are being called within Oracle as well as system calls. DStackProf, a tool created by Tanel Poder, see [18] for more resources, can be used on the Solaris operating system to analyse the stack traces of the Oracle processes.

Once identified with such a tool, measurements can be made on the time it takes for the system call to be processed, this can, for example, be done with the Solaris truss command.

4. Overloads

4.1. Overload at the disk / disk driver level

The disk subsystems are only capable of a certain number of random IO operations per second, each disk being capable of 100 to 250 random operations per second (depending on the spinning rotation speed and the interface used). It implies that, if more operations are given in the run queue, they will take longer to be processed. The figure 5 shows that if not enough load ("8 threads") is put on the system, the best performance cannot be achieved. With "16 threads" the maximum number of IO operations per second is achieved. With "32 threads" or "64 threads" threads, the run queue is too long and the time taken per IO is higher.

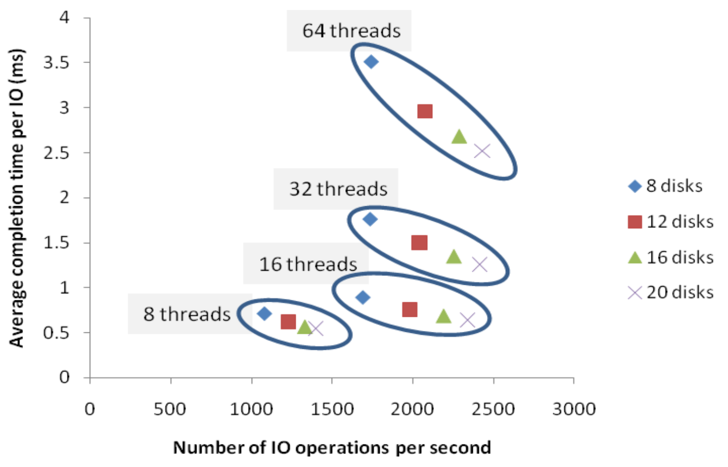


Figure 5. In this example, we show that overload at the disk level can lead to wrong interpretation of the Oracle instrumentation information.

4.2. Overload at the CPU level

We have often observed that there can be confusions regarding the source of performance problem. Indeed, Oracle, through the instrumentation layer, provides information about the timing required to perform the IO operation from the database session point of view. It may not represent the real time required to perform the IO operation. This is the case, for example, in case of CPU overload: the operating system may not be able to have the session process on the CPU right after the IO operation has been performed if many processes are active at the same time.

This can be illustrated as follows: in a system under acceptable load, when an Oracle session requires doing an IO operation, it records the first timer information, then makes some operations and calls the system call. The Operating System will perform some operations and perform the operation to the disk subsystem. Once the operation at the disk level is performed, it will return to the Operating System (OS) and then to the Oracle layer which will take the second timer information, the difference `t2-t1` being the recorded time, at the Oracle session level for the wait.

Under high load, the same logic applies but the operating system might not be able to schedule the Oracle process at the OS level rapidly after the IO operation has been performed, hence larger values for the `t2-t1` Oracle level delay for the wait. This is illustrated in figure 6.

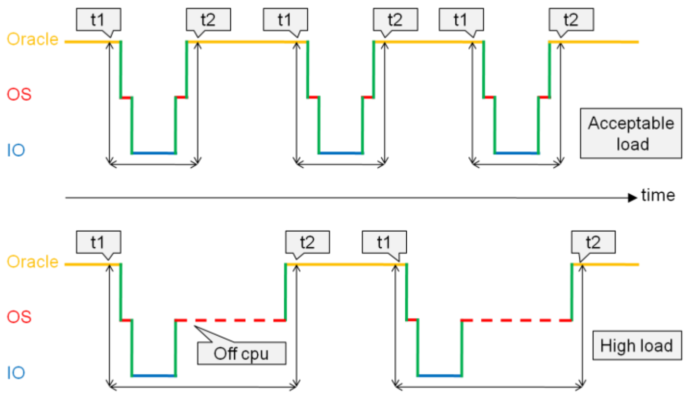


Figure 6. In this example, we show that overload at the CPU level can lead to wrong interpretation of the Oracle instrumentation information.

DTrace can be used to investigate the reason for the difference between normal load and high load with the exact same IO load. In figure 7, one can see that the distribution of the time spent for the process off-cpu. A small DTrace script, as shown in Appendix A can help to investigate such questions.

```
normal load - avg_cpu_off =  6768876 - off-cpu distribution
           value  ------------- Distribution ------- count
         1048576 |                                   2
         2097152 |@@@@                               86
         4194304 |@@@@@@@@@@@@@@@@@@@@@@@@@@@        577
         8388608 |@@@@@@@@@                          189
        16777216 |                                   2
high load  - avg_cpu_off = 10409057 - off-cpu distribution
           value  ------------- Distribution ------- count
           16384 |                                   4
           32768 |@                                  11
           65536 |                                   2
         2097152 |@                                  15
```

```
  4194304 |@@@@@@@@@@@@@@                       177
  8388608 |@@@@@@@@@@@@@@@@@@@@                 249
 16777216 |@@@                                  41
 33554432 |                                     4
```

Figure 7. off-cpu distribution under normal or high load with the same IO load.

5. Flash Solid-State Drive

Solid-State Drive devices are devices which emulate a hard disk drive interface even so it internally uses solid-state memory to store the data. They exist in several types, in this paper, we concentrate on the flash memory based ones and not the Random-access memory (RAM) based devices. There exist several types of flash SSDs, depending on the type of memory they use: single-level cell and multi-level cell. The single-level cell devices are faster and said to be more reliable, it is among these ones that some SSDs are called “enterprise SSDs”.

The interest in SSDs for database type workloads is in the ability of delivering a lot of random IO operations per second with short response time. In the recent years, we have seen the need to have large number of disks only for providing enough random IO operations resources. One of the benefits for SSD devices is that is that the number of IO operations per second per Watt is higher, thus for a system sized on the IO operations per second factor, it will require less power.

It is important to note that the competition in the consumer market is typically on laptop/desktop workload and not on the ability to process large number of IO operations per second and random write operations. As the consumer market is often a driving factor for the whole computing hardware industry, one has to, as normally done, validate the configuration based on the application workload. By nature, the flash based SSDs require additional capacity and complex algorithms for handling the write operations in a fast manner.

At CERN, a number of tests have been performed with Oracle database workload and a high variety of results has been identified depending on the device. For example figures 8 and 9 present distribution of the time taken by 8Kib random write operation on two different devices.

On the first device, one can see that the write operations are processed in average in 196µs with a standard deviation of 77µs, we can observe on the graph of figure 8 that few IO operations require more than 2ms to be processed. The device can capable of 4941 write operations per second sustained.

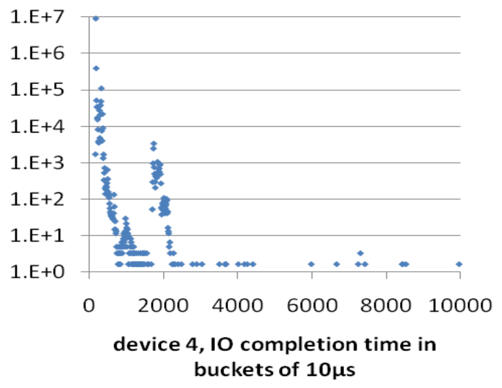


Figure 8. Distribution of write IO operations on device 4.

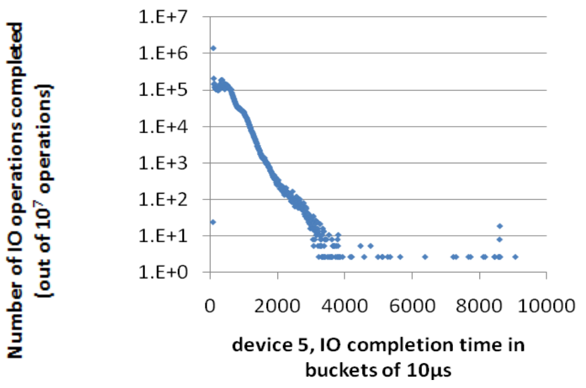


Figure 9. Distribution of write IO operations on device 5.

On the second device for which the results are presented, the average is also very low (but higher than for the first device): 528µs but the standard deviation is high: 2ms and indeed one can observe the fact on figure 9 that the distribution of the operation time is much wider than for the first device.

6. Conclusions

Although this paper concentrates on IO performance, target should be to limit the amount of physical IO required in the first place. When investigating IO related application performance problems, it is better to target the amount of logical IO operations as described in [19].

The different solutions deployed by the WLCG sites have been compared: ASM, cluster filesystems and NFS storage systems. One should note that these three solutions are both cluster enabled and offer a higher level view of the storage than traditional Oracle installation with raw devices. ASM is the most widely deployed solution. It has been shown that Oracle Exadata provides stability and performance for some very demanding workloads thanks to its offloading features. Exadata shows great promise and allows for high throughput data acquisition.

The results of this work demonstrate that new tools like DTrace and Active Session History provide database experts easier and non-intrusive ways to investigate IO issues compared with traditional tracing of the Oracle sessions. Two types of misunderstanding related to typical overload scenarios have also been demonstrated and explained.

Results of tests on flash SSD devices have been presented. They show important variation among devices for the completion time for write operations, from 77µs to 2ms as standard deviation. For databases, the write operations time to completion is very important as synchronous redo log write operation is part of the commit operation. One of the tested flash SSD devices can provide up to 17 times the number of random IO operations per second than the best hard drives. It implies that less power is required for some deployments. The flash SSD devices can be recommended either as primary storage (for redo log files and high IO requirements datafiles if such files are well identified and storage capacity permits) or as another level of read/write caches in other cases.

Acknowledgments

I would like to thank CERN's IT-Database and Engineering Services group for the help and support, Elzbieta Gajewska-Dendek and Anton Topurov for their work on the Swingbench PVSS workload implementation and the tests performed in the CERN openlab framework, Péter Kelemen for the help on the SSD tests and Oracle for the possibility to perform the tests on Exadata, especially Vincent Burguburu, Owen Hughes, Graeme Kerr, Bjørn Kisbye Engsig, Monica Marinucci and Michel Sitbon.

Appendix A : DTrace script to investigate IO response time under high load

```
syscall::pread:entry
/pid == $target && self->traceme == 0 /
{
  self->traceme = 1;
  self->on = timestamp;
  self->off= timestamp;
  self->io_start=timestamp;
}
syscall::pread:entry
/self->traceme == 1 /
{ self->io_start=timestamp; }
syscall::pread:return
/self->traceme == 1 /
{
  @avgs["avg_io"] = avg(timestamp-self->io_start);
  @[tid,"time_io"] = quantize(timestamp-self->io_start);
  @counts["count_io"] = count();
}
sched:::on-cpu
/pid == $target && self->traceme == 1 /
```

```
{
  self->on = timestamp;
  @[tid,"off-cpu"] = quantize(self->on - self->off);
  @totals["total_cpu_off"] = sum(self->on - self->off);
  @avgs["avg_cpu_off"] = avg (self->on - self->off);
  @counts["count_cpu_on"] = count();
}
sched:::off-cpu
/self->traceme == 1/
{
  self->off= timestamp;
  @totals["total_cpu_on"] = sum(self->off - self->on);
  @avgs["avg_cpu_on"] = avg(self->off - self->on);
  @[tid,"on-cpu"] = quantize(self->off - self->on);
  @counts["count_cpu_off"] = count();
}
tick-1sec
/i++ >= 5/
{  exit(0); }
```

References

[1] Oracle `http://www.oracle.com/`

[2] Schinzel J 1982 Oracle : The Database Management System For LEP CERN-LEP-Note-374

[3] CERN Engineering and Equipment Data Management Service `http://edms.cern.ch/`

[4] Shiers J 2006 The Evolution of Databases in HEP *15th Int. Conf. on Computing In High Energy and Nuclear Physics, Mumbai, India, 13 - 17 Feb 2006* pp 303-307

[5] Distributed Deployment of Databases for LCG `http://lcg3d.cern.ch/`

[6] Gonzalez-Berges M 2007 The High Performance Database Archiver for the LHC Experiments *ICALEPS'07, Knoxville, Tennessee USA, 2007* p 517

[7] WLCG, The Tier Sites `http://cern.ch/LCG/public/tiers.htm`

[8] Adams S Use asynchronous I/O `http://www.ixora.com.au/tips/use_asynchronous_io.htm`

[9] Closson K 2006 Analyzing Asynchronous I/O Support with Oracle10g `http://kevinclosson.wordpress.com/2006/12/06/analyzing-asynchronous-io-support-with-oracle10g/`

[10] Adams S Direct I/O `http://www.ixora.com.au/notes/direct_io.htm`

[11] Oracle Announcement on using Raw devices with release 11.2 MetaLink Note 754305.1

[12] M Girone 2008 CERN database services for the LHC computing grid *J. Phys.: Conf. Ser.* **119** 052017 (7pp)

[13] Raw Devices and Cluster Filesystems With Real Application Clusters MetaLink Note 183408.1

[14] M Guijarro and R Gaspar 2008 Experience and Lessons learnt from running High Availability Databases on Network Attached Storage *J. Phys.: Conf. Ser.* **119** 042015 (10pp)

[15] Swingbench `http://dominicgiles.com/swingbench.html`

[16] Millsap C 2009 For Developers: Making Friends with the Oracle database `http://method-r.com/downloads/cat_view/38-papers-and-articles`

[17] Wood G Sifting through the ASHes `http://www.oracle.com/technology/products/manageability/database/pdf/twp03/PPT_active_session_history.pdf`

[18] Poder T blog `http://blog.tanelpoder.com/`

[19] Millsap C 2001 Why You Should Focus on LIOs Instead of PIOs `http://www.hotsos.com/`