NetApp
Go further, faster

Technical Report

A Discussion of Fractured Block in the Context of Oracle Backup

Brandon Hoang, NetApp
June 2010 | TR-3850

TABLE OF CONTENTS

1 WHAT ARE FRACTURED BLOCKS?

An Oracle® block contains special header information and tailcheck, which allows Oracle to determine a discrepancy. Fractured block refers to a condition in which the head and the tail of an Oracle block do not match. A fractured block can occur if Oracle is updating a block at the same time that the copy utility (OS or storage based) is reading the same block. The copy utility could capture an image of a partially updated block. A fractured block situation, although extremely rare and timing dependent, can happen if the copy utility operates at an I/O-block size that is smaller or is not a multiple of the Oracle block size.

An Oracle block is the smallest unit of I/O performed by the Oracle instance. The Oracle block size is set during database creation and can range from 2kB to 32kB. An OS block size varies by platform, with most UNIX® and Linux® systems adopting a 4kB or 8kB block size. Windows® has a default size of 2kB or 4kB.

As an example, assume that the Oracle block size is 8kB and the OS block size is 4kB. Oracle is writing to the data files at 8kB block size, while the OS backup copy is reading at 4kB increments. The order in which the 4kB block is read is determined by the OS and is completely independent of Oracle. The OS copy could read the first or second 4kB half of the Oracle block just before the database writer (DBWR) updates the block. The block then gets updated with the new image. Subsequently, the copy reads the remaining first or second 4kB half of the block. The copy image of that particular block now contains two halves that are from different points in time, perhaps with different system change numbers (SCNs).The result is a fractured, or split, block.

A fractured block occurrence is exceptionally uncommon because it is not only timing dependent; it also requires that certain governing conditions be present. In fact, the governing conditions must first be present. If the timing is also right, only then can fractured blocks occur.

Fractured blocks can be prevented during backups by using Recovery Manager (RMAN) and backup mode options. RMAN never copies a fractured block. It knows the correct order to read the blocks and rereads the block until it gets a consistent image. Backup mode has contingent block image logging to recover from fractured blocks. In contrast, snapshot or backup copy of database created without backup mode does not have the provision to resolve fractured blocks. Fractured blocks can occur when the backup process is non-RMAN and the database is not placed in backup mode.

1.1 CAUSES OF FRACTURED BLOCKS

Fractured blocks have numerous causes. They can result from bugs or failures in hardware, software, or firmware; however, the principal cause of a fractured block is the OS or copy utility and Oracle operating at different I/O sizes, which could result in a partially copied Oracle block. For example, the OS could split a single Oracle block write into multiple smaller physical writes, or it could perform a read of an Oracle block in multiple OS reads. Such occurrences can lead to an Oracle block being partially written or read while the copy or snapshot is created. See Figure 1 for an illustration of how a fractured block can occur when Oracle writes are split.

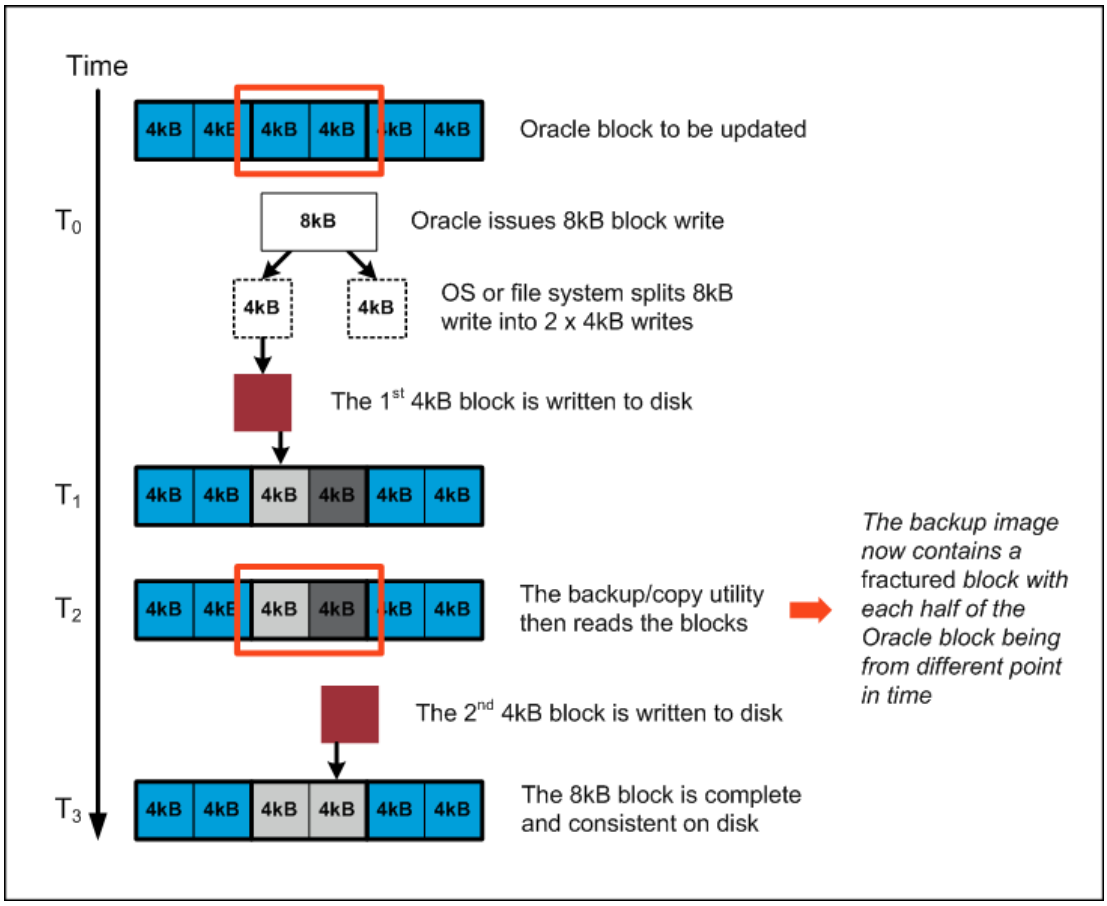

Figure 1) Fractured blocks can occur when Oracle writes are split.

Fractured blocks can occur due to one or more of the following conditions:

- OS, file system, or volume manager splits a single Oracle I/O write operation into multiple I/Os at a lower level
- Copy utility of the OS/file system/Logical Volume Manager (LVM) or storage system performs read operations at a block size smaller than that of the Oracle block size, resulting in multiple reads of a single Oracle block
- Possible failures or bugs on OS, LVM, hardware, host bus adapters (HBAs), or system, which causes an Oracle I/O write to be interrupted before completion

See Figure 2 for an illustration of a fractured block resulting from backup or copy utility reading at smaller block size than the block size at which Oracle is reading.

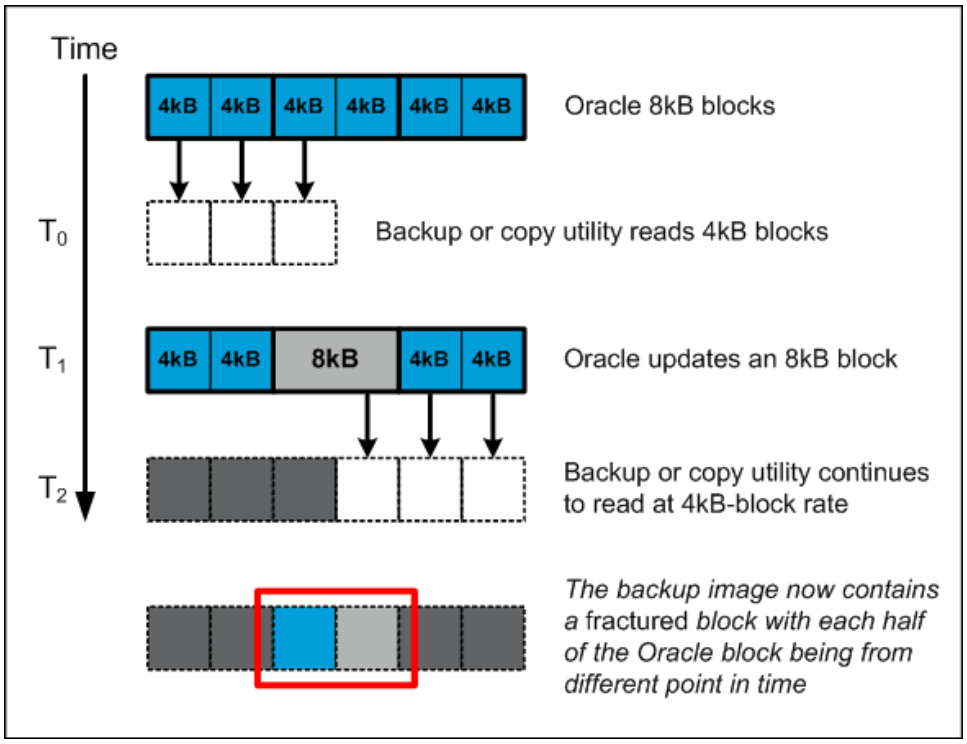

Figure 2) Fractured blocks can occur when backup/copy utility reads at smaller block size than that of Oracle.

2 AVOIDING FRACTURED BLOCKS

One way to avoid fractured blocks is to maintain atomicity of an Oracle block read/write. This means making sure a single Oracle block read/write gets processed as a single atomic I/O operation at the lower levels and is never split into multiple I/Os. The maximum I/O size handled by the OS, the file system, or the volume manager must always to be a multiple of the Oracle block size, and the Oracle block size must always be the minimum unit of I/O performed on data files by Oracle. Thus, the OS, file system, or volume manager, which are capable of issuing I/O sizes in multiples of Oracle block size, would not break a single Oracle block I/O into multiple operations.

2.1 DETERMINING THE MAXIMUM I/O SIZE BEYOND THE ORACLE LAYER

The structure of an I/O stack is usually defined by multiple layers and components. A Network File System (NFS) stack typically comprises the NFS client, TCP-IP protocol, and the network device drivers. A storage network area (SAN) environment with block-based protocol is more complex and most likely depends on additional layers, whereas a file system implementation may share a subset of components seen in a SAN configuration. See Figure 3.

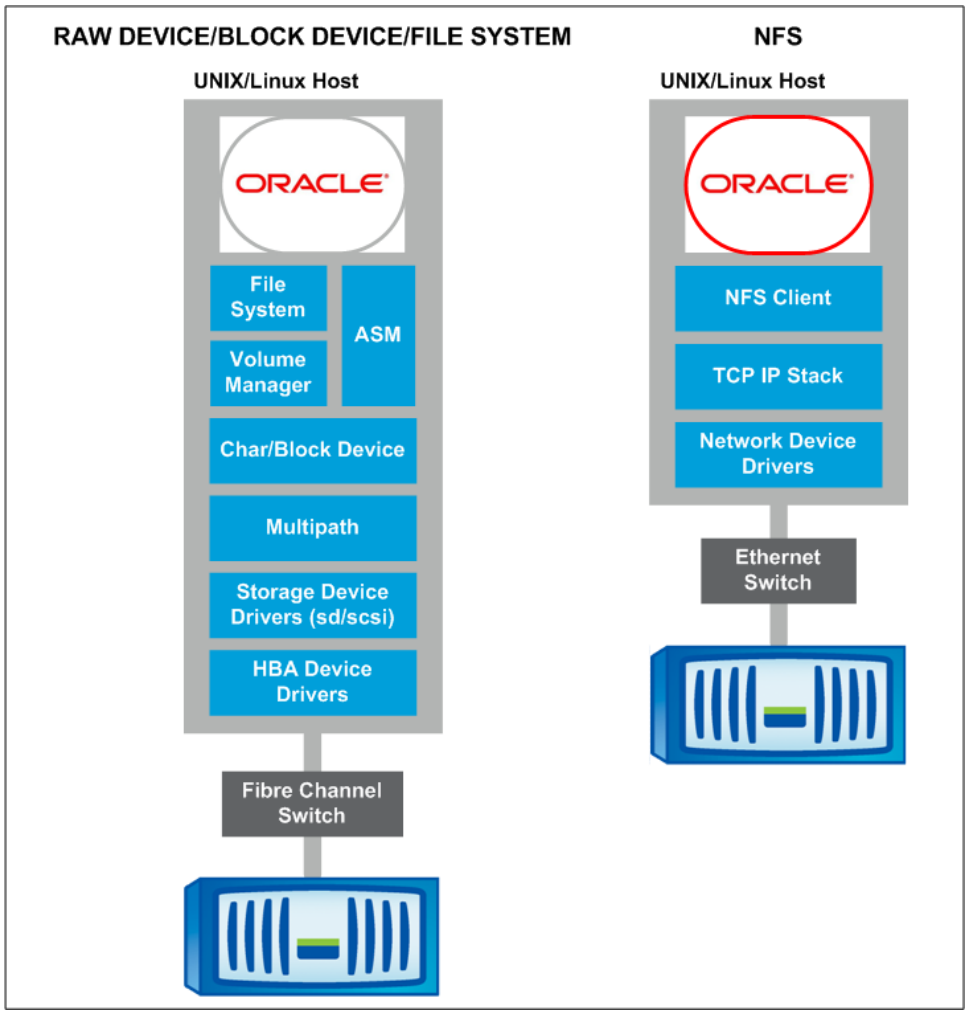

Figure 3) Typical I/O stack configuration of SAN and NAS environments.

Because the I/O stack is interconnected by different layers, the maximum I/O size or I/O request size can be independently determined or restricted by some of the layers. This behavior is more evident in a SAN configuration, where, for instance, the OS, the volume manager, and the device drivers have their own attributes that control the maximum I/O size or I/O request size. When the file system is used with buffered I/O and not direct I/O, the file system block size or buffer size, or OS page size may dictate the size of the I/O, which the OS uses to write or read files.

I/O subsystems can vary widely by OS, platforms, kernel releases, protocols, and/or file system types. The following sections contain details on the key characteristics of block device, file system, and NFS configurations.

2.2 RAW DEVICES AND BLOCK DEVICES

Raw devices, also known as character devices, were once commonly implemented in Oracle environments to improve I/O performance. Raw devices are typically addressed as `/dev/rdsk` or `/dev/raw`. Raw device is a special kind of block device that allows applications such as Oracle to access the underlying storage or disks directly without passing through the operating system's cache and buffers. Thus, the default mode for Oracle to access raw devices is direct I/O and asynchronous. Due to complex management and administration such as offset and alignment, use of raw device is becoming less preferred. In addition, raw device support is scheduled to be deprecated in some operating systems such as Linux.

Block devices, by definition, transfer data in fixed-size groups of bytes—block, sector, or cluster. Block devices generally operate with buffered I/O, where the operating system allocates data buffers to hold blocks for input and output. Reading and writing to block devices are buffered I/Os by default. However, since the Oracle 10*g* Release 2, Oracle takes advantage of the availability of the O_DIRECT feature by automatically opening all block devices using the O_DIRECT flag, allowing all I/Os to bypass the OS buffers.

FACTORS DETERMINING MAXIMUM I/O SIZE AND I/O REQUEST SIZE FOR RAW DEVICES AND BLOCK DEVICES

When operating with raw devices and block devices in a SAN environment, different layers in the I/O stack can impose a limit on the maximum I/O size. The following list contains descriptions of those layers:

Operating systems: Solaris™ and HP-UX use `maxphys` kernel variable to set the largest physical I/O request size. The `maxphys` kernel variable has a default of 128kB on Solaris and 1MB on HP-UX.

Linux offers the aio-max-size parameter in the 2.4 kernel releases, which specifies the maximum block size that one single AIO write/read can perform. Starting with Red Hat Enterprise Linux 4 and 5 (2.6 kernels), this option is no longer available and instead is automatic, based on the physical capabilities of the disk device driver.

Volume managers: Some volume managers, such as Veritas™ VxVM, have tunables that limit the size of I/O requests to the volume manager. The VxVM tunable `vxio:vol_maxio` defines the largest I/O request size that can be made to the volume manager without the system breaking up the I/O requests into the specified size. The default value of `vxio:vol_maxio` is usually 256kB.

Solaris Volume Manager/Disk Suite relies on the `md_maxphys` kernel parameter to limit the maximum I/O size at its layer. The default value is the same as the `maxphys` kernel parameter.

Device drivers: Most device drivers have a limit on the maximum I/O size that can be transferred. For example, Solaris has device drivers such as `sd` (SCSI device driver) and `ssd` (Fibre Channel device driver), which usually support an upper limit of 1MB I/O size.

In addition, Fibre Channel HBAs can support up to a certain maximum I/O request size. The maximum size (8 to 16MB) is based on the direct memory access (DMA) size allowed within the designed Fibre Channel chipset, which is usually much higher than the limits at the OS, volume manager, and device driver layers.

Note: Raw devices and block devices with direct I/O usually are not exposed to the potential of fractured blocks. I/Os performed on raw devices and unbuffered block devices are inherently direct. In addition, the maximum I/O size permitted by each layer (OS, volume managers, device drivers) is almost always much larger than the largest Oracle block size of 32kB. Thus, single Oracle block I/Os are not subject to being split into multiple I/Os by the various layers.

2.3 FILE SYSTEMS

File system is a popular option for Oracle database deployments. Compared to raw devices and block devices (without the use of Oracle Automated Storage Management [ASM]), a file system offers simpler management and administration, as well as backup and recovery.

Most file systems are based on block devices, which are usually constructed on hardware RAID volumes/LUNs or LVM volumes. By default, file systems employ buffered I/Os, where the operating system page cache is used to transfer data between the application and the storage device.

In file systems, there is the logical block size, which refers to the block size of the file system, and the physical block size, which is the actual operating system's physical block size. The physical block size is the smallest block size that the disk controller can read or write—512 bytes on most operating systems. The logical block size is the size of the blocks that the UNIX or Linux kernel uses to read or write files. The logical block size may vary by vendors' implementations and platforms, though it is typically a multiple of the physical block size, with the maximum size being the kernel page size.

The following list describes the various file systems:

- **Solaris UFS file system.** Supports block sizes of 4 or 8kB. The default logical block size is usually set to the page size of the system, 4kB on x86 and x86_64 architecture and 8kB on Sparc.
- **Veritas file system (VxFS).** Supports block size/extent of 1024, 2048, 4096, and 8192 bytes.

- **HP-UX JFS.** Is based on VxFS; supports block size of 1024, 2048, 4096, and 8192 bytes with the default being 1024 bytes.
- **AIX JFS/JFS2.** Supports block sizes of 512, 1024, 2048, and 4096 bytes.

FACTORS AFFECTING MAXIMUM I/O SIZE IN FILE SYSTEMS

When using buffered I/O, which is the default mode for file systems, some file systems may break up I/Os into a certain size. For example, the Solaris file system breaks synchronous writes into 8kB by default. JFS on HP-UX has a buffer size, `max_buf_data_size`, of 8kB (or 64kB), which determines the I/O size. VxFS with the Veritas Volume Manager (VxVM), by default, breaks up I/O requests that are larger than 256kB. In general, the file system buffer size dictates the maximum I/O size that can be performed when using buffered I/O mode.

For Oracle data files, the normal recommendation is to enable direct I/O with file systems. There are several reasons that direct I/O is preferred:

- Facilitate better overall I/O performance for Oracle
- Avoid breaking up I/Os in some file system implementations (hence allowing larger I/Os to complete as a single operation)
- Eliminate double caching of Oracle data by avoiding the use of the operating system buffer cache and memory for caching data
- Reduce CPU overhead as it eliminates the need to copy data to and from the file system buffer cache
- Reduce locking contention associated with managing OS memory

With direct I/O, I/O size allowed by the file systems is no longer restricted by the buffer size. By enabling direct I/O, Solaris UNIX File System (UFS) does not break writes into 8kB size; instead it allows large writes to complete as a single I/O. JFS has the maximum direct I/O size of 256kB versus the maximum buffered I/O size of 64kB. VxFS has the maximum size of a direct I/O request defined by `max_direct_iosz` tunable parameter. If a larger I/O request comes in, it gets broken up into `max_direct_iosz` chunks.

Most of the operating systems and file systems now offer direct I/O implementation. The way in which direct I/O is activated can vary by operating systems and file systems. Direct I/O can sometimes be enabled for an entire file system through the file system mount option or direct I/O is invoked at the file-handler level using direct I/O option during file access system calls.

Some examples of enabling direct I/O for Oracle under various operating systems and file systems:

- **Solaris:** UFS supports direct I/O through the `forcedirectio` mount option. This mount option is appropriate for enabling direct I/O if you are using Oracle 8*i*. Beginning with Oracle 9*i*, Oracle can use application programming interface (API) calls to enable direct I/O when opening files. Setting the Oracle init parameter, `filesystemio_options=setall`, is the preferred way for enabling direct I/O in 9*i* or later.
- **HP-UX:** Direct I/O on JFS is enabled by setting the `convosync=direct` mount option.
- **AIX:** Starting with AIX 5.1, direct I/O is enabled using the `dio` option on the mount command. This option works for both JFS and JFS2 file systems. Direct I/O can also be enabled at the file level in Oracle Database 10*g* and later. This is achieved by setting the Oracle FILESYSTEMIO_OPTIONS parameter to `setall` or `directio`.
- **Linux:** Direct I/O under Linux is available at the file level. Oracle's init parameter, `filesystemio_options`, should be set to `setall`, which implements both direct I/O and asynchronous I/O.
- **Veritas VxFS:** Mount option `convosync=direct` is used to enable direct I/O. This option is applicable when using VxFS on Solaris, HP-UX, or AIX. Veritas also offers Veritas Quick I/O, which overcomes the limitation of single writer lock implementation seen by POSIX-complaint direct I/O. Veritas ODM (Oracle Disk Manager) is another product with direct I/O feature.

Note: In general, to minimize the risk of fractured blocks occurring with file systems, avoid using buffered I/O and use direct I/O instead. In addition, configure the file system block size in multiples of the Oracle block size or the same as the Oracle block size when possible. OLTP databases generally use 8kB Oracle block size and most file systems support 8kB block size.

2.4 NETWORK FILE SYSTEM

In NFS, the `rsize` and `wsize` mount options specify the size of the data chunks that are exchanged between the client and the server. If no `rsize` and `wsize` options are specified, the default varies between versions of NFS. NFSv2 usually has a default of 4kB with the maximum size of 8kB for read/write operations. For NFSv3 over TCP, most platforms have a default of 32kB read or write operation. NetApp® platforms support 64kB for `wsize` and `rsize`.

When using Oracle, mounting a file system with a `wsize` value that is smaller than the Oracle block size can cause Oracle data block writes to be split into multiple data chunks, which can result in fractured block writes. Setting the value of the `rsize` mount option smaller than the database block size can also result in reading a block image that is partially updated and is, therefore, fractured.

To avoid the potential for fractured blocks, Oracle requires that NFS file systems be mounted with `rsize` and `wsize` that are greater than or equal to the database block size supported by Oracle. Oracle recommends a setting of 32768 for either `rsize` or `wsize.` (The source for these settings is "Metalink Note 359515.1 - Mount Options for Oracle files when used with NAS devices.")

As of Oracle 10.2, the I/O interface layer in Oracle has built-in checks to verify that the mount options for NFS-mounted file systems are set correctly. The exact checks on NFS file systems used by Oracle may vary among platforms, but the basic checks include:

- The mount table (for example, `/etc/mnttab`) can be read to check the mount options.
- The NFS file system is mounted with the hard option.
- The mount options include `rsize` >=32768 and `wsize` >=32768.
- For RAC environments, where NFS disks are supported, the `noac` mount option is specified.

Note: To avoid fractured blocks with NFS, use direct I/O by specifying mount options `forcedirectio` on Solaris and HPUX, or `cio` on AIX. Set the Oracle init parameter, `filesystemio_options`, to `setall` or `directio.` In addition, set `rsize` and `wsize` in multiples of Oracle block size, starting with a minimum value of `32768.`

2.5 DIRECT NFS (BY ORACLE)

Direct NFS is the Oracle NFS client, available in Oracle 11*g*. It is designed to integrate the NFS client functionality within the Oracle Relational Database Management System (RDBMS) server code. Direct NFS delivers several core benefits over the standard kernel NFS client—simplicity, scalability, high availability, load balancing, and I/O performance.

Direct NFS client uses the NFS ODM library to perform I/Os, which avoids the kernel NFS stack and eliminates write-ordering locks. By design, Direct NFS is capable of executing concurrent direct I/O and asynchronous I/O.

Direct NFS is Oracle's own design and architecture. Unlike kernel NFS, there is no longer a concern for making sure that the appropriate I/O size is set. Direct NFS has knowledge of the Oracle block size used by the database; thus, it guarantees that I/Os are performed in multiples of the Oracle block size.

3 NETAPP STORAGE AND I/O ATOMICITY

Database technologies are usually designed to deliver the strictest level of data integrity and consistency. In particular, transactional databases tend to enforce maximum consideration on transaction reliability. To make sure that transactions are managed and processed reliably, database design aims to adhere to a set of fundamental properties well referenced in the database industry, namely atomicity, consistency, isolation, and durability (ACID).

- **Atomicity:** Either all of the tasks in a transaction are performed or none is committed. This property demands an all-or-nothing approach. The transaction must be completed. If part of the transaction fails, the entire transaction must be rolled back.
- **Consistency:** The database must remain in a consistent state before the start of the transaction and after the transaction. Every transaction must preserve the database's consistency rules.

- **Isolation:** No transaction has knowledge of other active transactions and no transaction can access another transaction's intermediate data.
- **Durability:** Once completed, a transaction and its result must persist. Completed transactions cannot be aborted at a later stage.

In the context of fractured blocks, atomicity refers to the premise that a database block write must be either completely present on disk or completely absent from disk. Partially written blocks are not tolerated and usually require recovery. Oracle has the ability to detect fractured blocks by way of checksum mechanism or header and tail check.

Assuming that the OS and the host layer can preserve atomicity of an Oracle block write, that atomicity is maintained and guaranteed by NetApp storage systems as the data is passed to the storage systems and is committed to disk.

NetApp storage systems internally operate on 64kB chunks. Regardless of protocols and transport methods (Fibre Channel, NFS, or iSCSI), all read and write operations up to 64kB are processed atomically by WAFL®. I/O operations larger than 64kB are broken up at the 64kB boundary. Because an Oracle block size must be a power of two and ranges up to 32kB, any Oracle block read and write is atomic. Even a read or write failure is guaranteed to be atomic to 64kB.

While NetApp storage system processes data in 64kB chunks, the underlying WAFL file system is designed to manage data at 4kB block size; however, the 4kB block size does not represent the atomic boundary. Atomicity is still guaranteed up to 64kB. Any host writes up to 64kB either succeeds or fails; data associated with the writes are either fully committed on disk or completely absent.

The key point in the relationship between Data ONTAP® and WAFL as they relate to the potential of fractured blocks on Oracle is that NetApp storage system guarantees that I/O operations up to 64kB are atomic. All Oracle block reads or writes are managed correctly by NetApp storage systems. Fractured block occurrence is never the result of NetApp storage system.

When considering the potential of fractured blocks on Oracle, it is essential to recognize that NetApp storage system guarantees I/O operations up to 64kB to be atomic. NetApp storage systems correctly manage all Oracle block reads or writes; fractured block occurrence is never the result of NetApp storage system. For detailed information on the internal operation of Data ONTAP and WAFL, see the independent technical report, "On the Technical Suitability of Hosting Databases on Network Appliance Storage Systems" by Steve Daniel and Jeff Kimmel (http://media.netapp.com/documents/wp_3443.pdf).

4 HARDWARE-ASSISTED RESILIENT DATA AND NETAPP SNAPVALIDATOR

From both operational and backup standpoints, general block corruption and fractured blocks are a challenge in an Oracle environment. Several mechanisms and solutions are made available by Oracle to help detect and resolve block corruption. DB Block Checksum and FORCE LOGGING are direct features that can be enabled to prevent block corruption in an active instance. RMAN and hot backup mode are solutions to minimizing potential block corruption related to backup and recovery.

DB Block Checksum is the primary method for detecting corruptions that occur beyond the Oracle layer. It enables Oracle to detect block corruption that may be caused by software or hardware components associated with the I/O stack, such as file systems, volume managers, device drivers, HBAs, storage systems, or underlying disks. When DB Block Checksum is enabled (default in Oracle 9*i*) for each block that is modified, a checksum is calculated and stored in the block. When the block is accessed the next time, the checksum is validated to make sure that the block is not corrupt. DB Block Checksum detects block corruption at a certain level. However, because detection is triggered only upon the next read of the blocks, any blocks that are corrupted after leaving the Oracle layer are not prevented from being written to disk. After the blocks are passed down to the I/O stack (operating system, file systems, volume managers, and storage controllers), Oracle can no longer guarantee that the blocks being written to the storage systems are still correct. Therefore, Oracle alone cannot prevent corruption caused outside the Oracle instance from being written to permanent disk.

4.1 HARDWARE-ASSISTED RESILIENT DATA

The challenge described in the previous section has led Oracle to develop an initiative with storage vendors to provide external validation mechanism for Oracle data. The program is known as Hardware-Assisted Resilient Data (HARD). It requires participating vendors to incorporate equivalent checksum algorithms in their solutions to verify block integrity before the blocks are committed to disk, thereby preventing corrupted blocks from being written to disk.

To take advantage of HARD, Oracle data must reside on HARD-compliant storage systems, and the HARD validation feature must be enabled. All Oracle writes are validated within the storage system. If any incoming writes appear to be corrupt, the storage system rejects the writes to protect the integrity of the data. As a result, corrupted data blocks are never written to disks.

Using the HARD program, storage vendors can prevent the following types of corruption:

- Writes of corrupted blocks
- Writes of blocks to incorrect locations
- Erroneous writes by programs other than Oracle-to-Oracle data
- Corrupted third-party backups

Note: HARD is supported only through Oracle 10*g* release. At the time that this paper was published, Oracle 11*g* release was available but it did not support HARD. In collaboration with Emulex, Oracle is working on T10DIF Data Integrity extension, a standard for complete end-to-end data integrity checking for enterprise storage systems, which has been accepted into the 2.6.27 Linux kernel. It includes generic support for data integrity at the block and files system layers, as well as support for the T10 Protection Information Model and the Data Integrity Extensions. For more information on the progress of T10DIF, see the following Oracle Web site: http://oss.oracle.com/projects/data-integrity/documentation/.

4.2 NETAPP SNAPVALIDATOR

SnapValidator® is the NetApp implementation of the Oracle HARD initiative, a feature designed to leverage the Oracle checksum algorithms to detect and prevent data corruption before they arrive on disk.

By implementing Oracle data validation algorithms inside NetApp storage devices, NetApp can prevent corrupted data from being written to permanent storage. SnapValidator, coupled with Oracle DB Checksum, delivers end-to-end checksum functionality. It is also tightly integrated with the Oracle database architecture and complies with the Oracle HARD initiative. NetApp SnapValidator offers the following key benefits:

- Increased data reliability through end-to-end data protection
- Improved operational efficiency and avoided costs
- Modular accessibility
- Multiprotocol support

SnapValidator for Oracle is a Data ONTAP WAFL enhancement that allows NetApp storage systems to validate Oracle data when it is being written to the storage. With SnapValidator enabled, NetApp storage systems can stop data corruptions caused by other components within the I/O path from being written to permanent storage.

The following list briefly describes the internal operation of SnapValidator:

- Oracle block to be written is encoded with Oracle checksum by Oracle server.
- Data is passed through the I/O stack to NetApp storage.
- Once data reaches NetApp storage system and before it is written to disk, NetApp storage system performs a validation based on the following set of actions:
  - Checksum and verification against the data being sent
  - Verification of Oracle block number against the block the data is getting written into
  - Verification of length of write and the intended write size
  - Oracle magic number check
- If all verifications succeed, the data is then committed to disk; otherwise, NetApp storage system rejects the data and reports the error to the Oracle server.

Note: Because SnapValidator is a product of the HARD initiative, it supports releases only through Oracle 10*g*.

\

www.netapp.com