Proactively Preventing Data Corruption

Introduction

Data corruption is an insidious problem in storage. There are many types of corruption and many means to prevent them.

Enterprise class servers use error checking and correcting caches and memory to protect against single and double bit errors. System buses have similar protective measures such as parity. Communications going over the network are protected by checksums.

On the storage side many installations employ RAID (Redundant Array of Inexpensive Disks) technology to protect against disk failure. In the case of hardware RAID the array firmware will often use advanced checksumming techniques and media scrubbing to detect and potentially correct errors. The disk drives themselves also feature sophisticated error corrective measures, and storage protocols such as Fibre Channel and iSCSI feature a *Cyclic Redundancy Check* (CRC) that guards against data corruption on the wire.

At the top of the I/O stack, modern filesystems such as Oracle's *btrfs* use checksumming techniques on both data and filesystem metadata. This allows the filesystem code to detect data that has gone bad either on disk or in transit. The filesystem can then take corrective action, fail the I/O request or notify the user.

A common trait in most of the existing protective measures is that they work in their own isolated domains or at best between two adjacent nodes in the I/O path. There has been no common method for ensuring true end-to-end data integrity. . . until now. Before describing this new technology in detail, let us take a look at how data corruption is handled by currently shipping products.

1 Data Corruption

Corruption can occur as a result of bugs in both software and hardware. A common failure scenario involves incorrect buffers being written to disk, often clobbering good data.

This latent type of corruption can go undetected for a long period of time. It may take months before the application attempts to reread the data from disk, at which point the good data may have been lost forever. Short backup cycles may even have caused all intact copies of the data to be overwritten.

A crucial weapon in preventing this type of error is proactive data integrity protection: a method that prevents corrupted I/O requests from being written to disk.

For several years Oracle has offered a technology called HARD (Hardware Assisted

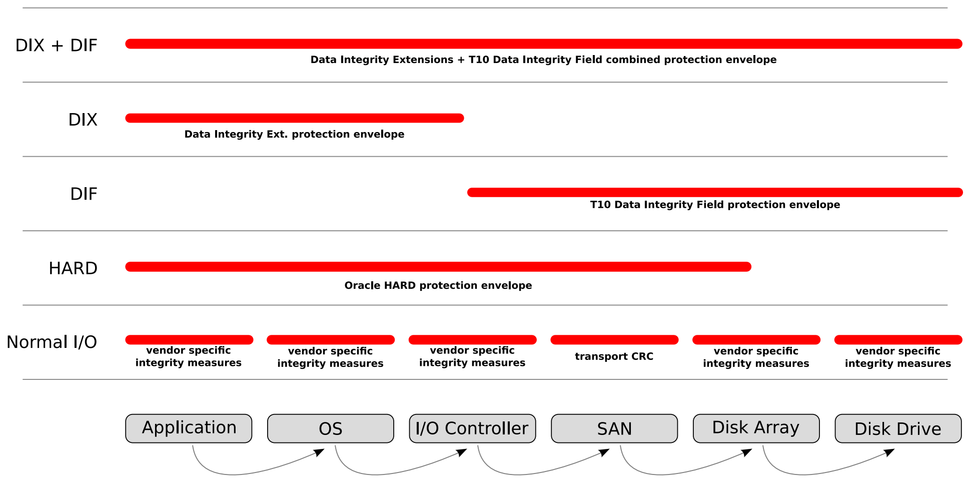
DIX + DIF
Data Integrity Extensions + T10 Data Integrity Field combined protection envelope
DIX
Data Integrity Ext. protection envelope
DIF
T10 Data Integrity Field protection envelope
HARD
Oracle HARD protection envelope
Normal I/O
vendor specific integrity measures
vendor specific integrity measures
vendor specific integrity measures
transport CRC
vendor specific integrity measures
vendor specific integrity measures
Application
OS
I/O Controller
SAN
Disk Array
Disk Drive

Figure 1 Five different protection scenarios. Starting at the bottom, the "Normal I/O" line illustrates the disjoint integrity coverage offered using a current operating system and standard hardware. The "HARD" line shows the protection envelope offered by the Oracle Database accessing a disk array with HARD capability. "DIF" shows coverage using the integrity portions of the SCSI protocol. "DIX" shows the coverage offered by the Data Integrity Extensions. And finally at the top, "DIX + DIF" illustrates the full coverage provided by the Data Integrity Extensions in combination with T10 DIF.

Resilient Data), which allows storage systems to verify the integrity of an Oracle database logical block before it is committed to stable storage. Though the level of protection offered by HARD is mandatory in numerous enterprise and government deployments, adoption outside the mission-critical business segment has been slow. The disk array vendors that license and implement the HARD technology only offer it in their very high end products. As a result, Oracle has been looking to provide a comparable level of resiliency using an open and standards-based approach.

A recent extension to the SCSI family of protocols allows extra protective measures, including a checksum, to be included in an I/O request. This appended data is referred to as *integrity metadata* or *protection information*.

Unfortunately, the SCSI protection envelope only covers the path between the I/O controller and the storage device. To remedy this, Oracle and a few select industry partners have collaborated to design a method of exposing the data integrity features to the operating system. This technology, known as the *Data Integrity Extensions*, allows the operating system—and even applications such as the Oracle Database—to generate protection data that will be verified as the request goes through the entire

I/O stack. See Figure 1 for an illustration of the integrity coverage provided by the technologies described above.

2 T10 Data Integrity Field

T10 is the INCITS standards body responsible for the SCSI family of protocols. Data corruption has been a known problem in the storage industry for years and T10 has provided the means to prevent it by extending the SCSI protocol to allow integrity metadata to be included in an I/O request. The extension to the SCSI block device protocol is called the *Data Integrity Field* or DIF.

Normal SCSI disks use a hardware sector size of 512 bytes. However, when used inside disk arrays the drives are often reformatted to a bigger sector size of 520 or 528 bytes. The operating system is only exposed to the usual 512 bytes of data. The extra 8 or 16 bytes in each sector are used internally by the array firmware for integrity checks.

DIF is similar in the sense that the storage device must be reformatted to 520 byte sectors. The main difference between DIF and proprietary array firmware is that the format of the extra 8 bytes of information per sector is well defined as well as being an open standard. This means that every node in the I/O path can participate in generating and verifying the integrity metadata.

Each DIF tuple is split up into three sections called tags. There is a 16-bit guard tag, a 16-bit application tag and a 32-bit reference tag.

The DIF specification lists several types of protection. Each of these protection types defines the contents of the three tag fields in the DIF tuple. The guard tag contains a 16-bit CRC of the 512 bytes of data in the sector. The application tag is for use by the application or operating system, and finally the reference tag is used to ensure ordering of the individual portions of the I/O request. The reference tag varies depending on protection type. The most common of these is Type 1 in which the reference tag needs to match the 32 lower bits of the target sector logical block address. This helps prevent misdirected writes, a common corruption error where data is written to the wrong place on disk.

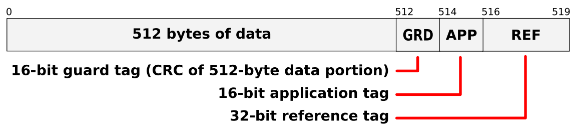

Figure 2 DIF tuple contents

If the storage device detects a mismatch between the data and the integrity metadata the I/O will be rejected before it is written to disk. Also, since each node in the I/O path is free to inspect and verify the integrity metadata, it is possible to isolate points of error. For instance, it is conceivable that in the future advanced fabric switches will be able to verify the integrity as data flows through the Storage Area Network.

The fact that a storage device is formatted using the DIF protection scheme is transparent to the operating system. In the case of a write request the I/O controller will receive a number of 512-byte buffers from the operating system and proceed to generate and append the appropriate 8 bytes of protection information to each sector. Upon receiving the request the SCSI disk will verify that the data matches the included integrity metadata. In the case of a mismatch, the I/O will be rejected and an error returned to the operating system.

Similarly, in the case of a read request the storage device will include the protection information and send 520 byte sectors to the I/O controller. The controller will verify the integrity of the I/O, strip off the protection data and return 512 byte data buffers to the operating system.

In other words, the added level of protection between controller and storage device is completely transparent to the operating system. Unfortunately, this also means the operating system is unable to participate in the integrity verification process. This is where the Data Integrity Extensions come in.

Allows I/O controller and storage device to exchange protection information.

Each data sector is protected by an 8-byte integrity tuple

The contents of this tuple include a checksum and an incrementing counter that ensures the I/O is intact.

Both I/O controller and storage device can detect and reject corrupted requests.

Figure 3 T10 Data Integrity Field

3 Data Integrity Extensions

While increased resilience against errors between controller and storage device is an improvement, the design goal was to enable true end-to-end data integrity protection. An obvious approach was to expose the DIF information above the I/O controller level and let the operating system gain access to the integrity metadata.

3.1 Buffer Separation

Exposing the 520-byte sectors to the operating system is problematic, however. Internally, operating systems generally work with sizes that are multiples of 512. On x86 and x86_64 hardware the system page size is 4096 KB. This means that 8 sectors fit nicely in a page. It is extremely inconvenient for the operating system to deal with buffers that are multiples of 520 bytes.

The Data Integrity Extensions allow the operating system to gain access to the DIF contents without changing its internal buffer size. This is achieved by separating the data buffers and the integrity metadata buffers. The controller firmware will interleave the data and integrity buffers on write and split them on read.

Separating the data from the integrity metadata in the operating system also reduces the risk of data corruption. Now two buffers in different locations need to match up for an I/O request to be successfully completed.

3.2 Performance Implications

DIF protection between I/O controller and disk is handled by custom hardware at near zero performance penalty. For true end-to-end data integrity, however, the application or the operating system needs to generate the protection information. Calculating the checksum in software obviously comes at a performance penalty and the T10 DIF standard mandates a heavyweight 16-bit CRC algorithm for the guard tag.

This CRC is quite expensive to calculate compared to other commonly used checksums. To alleviate the impact on system performance the TCP/IP checksum algorithm is used instead. This results in an almost negligible impact on system performance. The Data Integrity Extensions allow this alternate checksum type to be used by the operating system. The I/O controller will convert the IP checksum to the DIF CRC when sending a request to the storage device and vice versa.

The net result is that a full end-to-end protection envelope can be provided at a very low cost in terms of processing overhead.

4 Linux Data Integrity Framework

Oracle has implemented support for DIF and the I/O Controller Data Integrity Extensions in the Linux kernel. The changes are released under the GNU General Public License and have been submitted for inclusion in the official kernel tree. With this, Linux becomes the first operating system to gain true end-to-end data integrity protection.

Allow transfer of data integrity information to and from the host operating system.

Allow separation of data and integrity metadata buffers.

Allow a lightweight checksum algorithm to limit impact on operating system performance.

Figure 4 Data Integrity Extensions

The Linux changes allow integrity metadata to be generated and passed through the I/O stack. Currently the extensions are only accessible from within the kernel, but a userland API is in development. The goal is for all applications to be able to benefit from the extra data protection features.

Allows integrity metadata to be attached to an I/O request.

Allows integrity metadata to be generated automatically for unmodified applications.

Will allow advanced applications to manually send and receive integrity metadata.

Figure 5 Linux Data Integrity Framework

5 Future Developments

At a recent storage networking industry conference Oracle and its partners demonstrated an (unmodified) Oracle Database running on Linux using the data integrity framework. The server used a prototype Emulex fibre channel controller, a disk tray from LSI and disk drives from Seagate. We demonstrated how errors could be injected into the system, identified, isolated and remedied without causing downtime or on-disk corruption.

The SCSI standard only governs communications between I/O controller and storage device, and as such the interface between I/O controller and the operating system is outside the scope of the T10 organization. Consequently, Oracle and its partners have approached the Storage Networking Industry Association and set up Data Integrity Task Force with the intent to standardize the data integrity interfaces for applications, operating systems and I/O controllers.

Hardware products supporting DIF and the Data Integrity Extensions are scheduled for release in 2008.

Author

Martin K. Petersen has been involved in Linux development since the early nineties. He has worked on PA-RISC and IA-64 Linux ports for HP as well as the XFS filesystem and the Altix kernel for SGI. Martin works in Oracle's Linux Kernel Engineering group where he focuses on enterprise storage technologies. He can be reached at martin.petersen@oracle.com.