Using Logical Data Protection and Recovery to Improve Data Availability

Wei Hu

Oracle Corporation, 400 Oracle Parkway, Redwood Shores, CA 94065, USA
wei.hu@oracle.com

Abstract. Data availability is crucial to overall application availability. This paper describes data failures and classifies them as either physical or logical. Physical data failures such as data loss and corruptions are introduced in the I/O layers; logical data failures are introduced in the application layer. Current data protection techniques are geared towards physical data failures. This paper reviews physical data protection techniques and their limitations. It then introduces the concept of logical data protection and shows how applications such as the Oracle database can use application knowledge to implement logical data protection and recovery that are more effective than conventional physical data availability technologies.

1 Data Availability

An important aspect of high availability is data availability. *Data availability* is the extent to which an application's data is available and correct – i.e., in a form that allows the application to function. Data is not available when it is destroyed, inaccessible, lost, physically corrupted, or logically corrupted. An application cannot function without its data; highly available systems must therefore address *data failures*. Data failures (or unavailability) can be caused by hardware failure, software failure, human error (which includes malicious attack), and site failure.

Hardware failure can cause data loss and data corruption. Disk failures can cause data loss. Data corruptions can occur when the hardware incorrectly modifies data that is written or read back. Data corruption can be caused by faulty hardware and firmware. Uncorrected memory errors can corrupt the I/O buffers. Even faulty cables can corrupt the data that is transferred. In addition to *improper modification of data*, other variations of corruptions are: *misdirected write* where a data block is written to the wrong location and *lost write* whereby the storage subsystem acknowledged the completion of a write that was actually not done.

Software failure can also cause data loss and data corruption. Every data failure that is caused by hardware can also be caused by software. For example, bugs in file system and I/O code can lose writes, overwrite parts of the data that is being written, or write data to the wrong location. Another example of a software corruption is the *stray write* in which data structures in memory are overwritten by another thread. As the amount and complexity of software in the I/O subsystem grows, the probability of software-induced data failures also increases.

M. Malek, E. Nette, and N. Suri (Eds.): ISAS 2005, LNCS 3694, pp. 39 – 51, 2005.

Human error can account for up to 40% of unplanned application downtime [1]. Human error can cause all the data loss and corruptions caused by hardware or software. For example, a configuration error such as setting up a swap file over the same disk partitions that hold application data can cause data to be overwritten. This has the same effect as a corruption caused by misdirected writes.

A *site failure* is when an entire data center is lost. When a site disaster occurs, all data storage within a given geography is lost. This means that only data availability techniques that are geographically separated are effective against this type of data failure.

Even though data failures are relatively rare, they usually result in extended outage. The outage due to data failures is usually much longer than that caused by machine or process failure. Hours or even days of downtime are possible [2]. Applications that need high availability must therefore address data failures.

2 Physical and Logical Data Failures

We characterize data failures as either physical or logical based on where the data failure was introduced. We make this distinction because there are many solutions that only address physical data failures. The following figure shows how we model an application stack and the type of data failures.

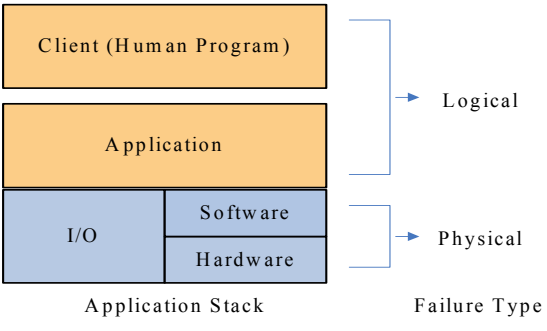

Fig. 1. Logical and Physical Data Failures

The I/O layer includes the hardware (adapter cards, cable, disk array) and software (file system, volume manager, and device driver). Applications such as databases use the I/O layer services. The client is a human or program that uses the services provided by the application layer.

A *physical data failure* is a data loss or data corruption that is introduced in the I/O software or hardware. A *logical data failure* is a data loss or data corruption that is introduced above the I/O software or hardware layers. We draw the boundary between physical and logical at the interface between the application and the I/O system because we want to describe the layering as seen from the application.

Note that a failure in a layer is not necessarily caused by code in that layer. A logical data corruption, for example, does not have to be caused by an application bug, it just has to be *introduced* while we are executing application layer code. For example, a memory fault that corrupts in-memory buffer used by the application would still cause a logical corruption even though the cause is hardware.

A logical corruption cannot be detected by I/O layer checks. This is because the data was already corrupted in the application layer by the time the data was passed to the I/O layer. The I/O system has no way to distinguish valid data from invalid data. More generally, a layer cannot detect errors introduced by layers above. An application, for example, cannot detect human errors. If a database administrator (client layer) were to issue a drop table command, the database (application layer) cannot always determine whether the human specified the correct table.

3 Physical Data Availability Techniques

In this section, we analyze technologies available for addressing physical data failures. Collectively, these techniques provide *physical data availability*. We start by discussing techniques for preventing physical corruptions, then techniques for recovering from physical data failure and site disaster.

Failure	Solution
Data Loss	RAID Backup/Restore Checksum Remote Snapshot
Data Corruption	Backup/Restore Snapshots Continuous Backup
Site Disaster	Offsite Backup Remote Mirroring

Fig. 2. Techniques for Physical Data Availability

3.1 Preventing Physical Data Failure

The primary means for preventing physical data loss is through redundant storage such as RAID [3]. Cyclic redundancy checksums (CRC) are also used to detect corruptions and error correcting code (ECC) memory is used to detect and repair errors [4]. These techniques have greatly improved the reliability of the I/O subsystem and reduced the frequency of data failures.

3.2 Recovering from Physical Data Failure

Backup/restore is the fundamental technique for recovering from physical data failures. When a data failure occurs, a backup of the data is restored. The limitations of using backups are data loss and restore time. After a backup has been restored, all the changes that were made since the backup was taken are lost. The exception is with applications such as databases that can use the transaction log to bring the backup to the current time. Restore time is an issue because a backup must first be restored from backup media before it can be used. Even with the use of on-disk backups instead of tape, restore time for large amounts of data can be lengthy.

To minimize data loss in the event of a restore, it is desirable to have frequent backups. Many storage vendors have optimized periodic backups so that they can be taken rapidly with relatively small disk overhead. These periodic backups are sometimes known as *snapshots* [5]. Snapshots can consist of full copies of the data (e.g., a mirror within the storage array), or a partial copy that tracks only changes since the last snapshot. Snapshots allow you to maintain multiple *restore points* to which you can bring the data in the case of a physical data failure. To address the issue of data loss after a restore, several *continuous backup* products [6,7] have recently been introduced. These apply database like logging techniques to track changes incrementally. This then allows you to return the data to any arbitrary point in the past.

3.3 Handling Site Disaster

Storing backup copies of the data off-site is a way to ensure that the data can be restored when the primary copy is unavailable. Remote disk mirroring can also be done so that one of the mirror copies is a remote storage unit. The remote copy is typically located in a data center that is not likely to be subject to the same disaster as the primary site. For example, it might be across a continent or an ocean. The geographic separation reduces the likelihood that a disaster that affects the primary site would also affect the backup site(s).

Note that remote mirroring over long distances can degrade the response time of the application because of propagation delays across long distances. In addition, for disaster recovery, all the other components of the application (application, hardware, networking) must also be available at the backup data center.

3.4 Limitations of Physical Data Protection Techniques

Physical data protection techniques provide protection against physical data failures. However, they only provide *partial* protection against logical data failures. In particular, physical data protection techniques can recover from, but cannot prevent, logical data failures.

3.4.1 Physical Techniques Cannot Prevent Logical Data Failures

Physical data protection techniques cannot *detect or prevent* logical data corruptions. This is because techniques such as RAID and checksums are physical checks. They cannot validate the contents of the data. So if a logically corrupted block were written

to a RAID device, the storage cannot detect that the data is invalid. This limitation applies to all physical data protection techniques. For example, a storage array that is remotely mirrored would ensure that a logically corruption would propagate bit-for-bit to all remote mirrors. This is a serious limitation given the wide range of potential logical data failures.

3.4.2 Physical Techniques Cannot Prevent Upper Layer Physical Data Failures

Significantly, physical data protection techniques cannot even prevent physical corruptions that were introduced at a higher layer in the I/O system. Modern I/O subsystems have become very complex. There are usually many layers of software, firmware, and hardware that is involved in an I/O request. Some of these components include: file system buffer cache, volume manager, device driver, host bus adapter, storage area network switch, storage controller, and the hardware/software inside the storage unit. A bug or failure in any of these components can cause a physical corruption that would not be detected by lower layers of the I/O subsystem. For example, RAID cannot correct a directory entry that has been corrupted by a file system bug. From the perspective of the RAID device, the file system error is also a logical failure.

3.4.3 Physical Techniques Are Not Optimal for Recovering from Logical Data Failures

Physical data protection techniques such as backup/restore and snapshots can be applied to recover from logical data failure. So for example, if a file was overwritten or accidentally deleted, a copy can be restored from backup. Physical data recovery techniques, however, have many limitations. First, the recovery is done at the granularity of the physical object (file, file system, disk, etc.). If only a subset of the data were damaged, physical recovery cannot repair just the damaged part, it has to restore the entire object. So if a database file were a terabyte in size and only a single row was corrupted, a physical restore would bring back the entire terabyte; it cannot restore just the row.

Doing file- or device-based recovery is not optimal because recovery takes longer and you will lose more data. Recovery takes longer since you are usually restoring more data than you need. Restoring an object typically means loss of data because the backup does not contain changes made since the backup was taken. Because the granularity is at the physical file or device, the data loss is also larger than required. For example, restoring the entire file means that you will lose changes made to all the records in the file even though only a single record is damaged.

In summary, physical data protection techniques are inadequate for applications that require high levels of data availability.

4 Logical Data Availability Techniques

We now look at how logical data protection techniques can address the limitations associated with physical data protection techniques. Logical data protection techniques exploit application knowledge to offer better protection against data

failures and better recovery from data failures. We will use the Oracle database as an example and show how it exploits knowledge of the logical structure of data to provide improved data availability. Figure 3 shows some of the solutions that we'll cover in this section.

Objective	Solution
Preventing Logical Data Failure	End-to-End Application Checksum Standby Database
Data Corruption	Backup/Recovery Flashback Precision Repair

Fig. 3. Techniques for Logical Data Availability

4.1 Preventing Logical Data Failure

The main logical data failure prevention techniques are application level checksums and logical replication.

4.1.1 End-to-End Application Checksum

Physical data protection technologies such as RAID and checksums are limited because they only validate the data when it is in the I/O subsystem. These techniques are ineffective against data that were corrupted prior to the initiation of the I/O request.

Applications can achieve higher data availability by implementing their own application level checksums. For example, when Oracle writes data, it performs a series of logical and physical checks on the data to ensure integrity. As part of this, it also stores checksum and other validation information in the data blocks. When the data is read back, Oracle verifies the checksum and other validation information in the blocks. If the block fails the checks, an error is raised. This detects I/O corruptions and prevents bad data from being used. Microsoft Exchange Server has a similar capability [8].

Conventional application level checksums can detect a corruption after the fact. Oracle's HARD (Hardware Assisted Resilient Data) [9] takes this further by preventing I/O corruptions from making it to disk in the first place. Figure 4 shows how this technology extends the range of validation all the way from the application (the database) to the hardware.

Under the HARD initiative, Oracle worked with storage vendors to imbed knowledge of Oracle block formats and checksums within storage arrays. When Oracle data is written to a HARD-compliant storage device, the storage device can

independently validate the integrity of the Oracle blocks as well as the locations to which the blocks were destined. If the validation checks pass, the data is written to the disks. If the validation checks fail, the write is not performed and an error is returned. This ensures that corrupted data are never written to disk.

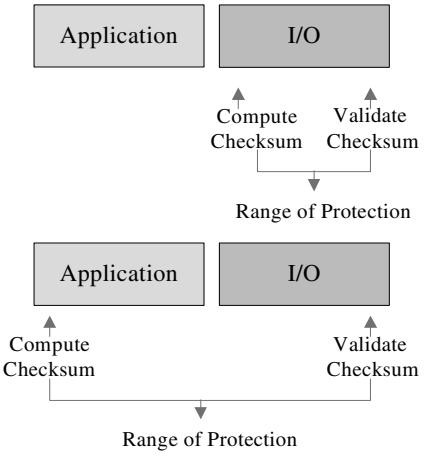

Fig. 4. End-to-end Application Checksum

The end-to-end application checksum can prevent a whole class of data failures. It protects against data corruptions introduced by any subsystem between the application (Oracle) and the disk array, it prevents misdirected writes, it also prevents overwrites of Oracle files by other applications.

The end-to-end application checksum also protects data during direct disk-to-tape transfers. Certain disk arrays can directly transfer data to tape drives without host intervention. This is useful for high performance backups. A corruption in this case is very severe as it means that the backups that you'll need to recover from other data failures would be corrupted. If the tape device is HARD compliant, it can directly validate the integrity of the data that it is receiving.

4.1.2 Standby Database

Application-maintained replicas, such as Oracle Data Guard [10,11] and Sybase Replication Server [12], are another powerful means for protecting data against both physical and logical failures. A standby database is a copy of the database that is kept up-to-date with changes as they are made on the primary database. Figure 5 shows how it works in a 2 database configuration:

1. A change is made on the primary database
2. The change is captured in the redo log
3. The log is shipped to the standby database
4. The change described in the log is applied to the standby database

If the primary database fails, the standby database can become the primary and continue operation. A primary may have multiple standby databases, each of which can be at a different location. A standby database that is geographically remote from the primary database can also offer protection against site failure.

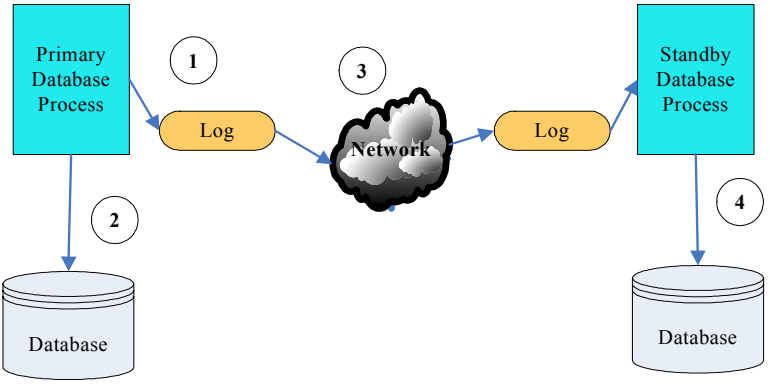

Fig. 5. Standby Database Operation

A standby database can be thought of as a remote mirror. As such, it can protect against the same physical data failures as storage-level remote mirroring. A standby, however, is more resilient against logical corruptions. This is because of how changes are propagated and applied at the standby databases. A storage-level remote mirroring solution sends description of the physical blocks that are changed. So if a logically corrupted block were written to the primary mirror, the remote mirroring software/firmware will ensure that the corruption is replicated bit-for-bit at the remote mirror.

Oracle Data Guard, on the other hand, receives a logical description of the change from the primary database. The redo log contains a great deal of contextual information such that if the log is corrupted or if the blocks being modified are inconsistent with the redo log record, the change application (described in step 4 above) would fail. This prevents many logical corruptions from propagating and corrupting the copy of the data on the standby.

4.1.3 Improving the Effectiveness of a Standby Using an Apply Lag

One can improve a standby database's effectiveness by building in an *apply lag*. In this mode, changes are still shipped immediately to the standby; however, the changes are not applied until after a configuration delay. Note that the lag does not mean that the standby will lose data after a failover. This is because there is no delay on receiving the changes at the standby, just on when these changes are applied. Since all the changes are available on the standby database, no data would be lost if the primary database were to fail.

The lag gives the customer a chance to react to human error or application corruption before the standby database is affected. For example, if one accidentally deleted (dropped) a table on the primary database, the delay means that the operation

would not be immediately carried out at the standby. This delay gives the system a chance to discover the error and immediately stop the standby apply. You can then apply the changes up to the point of the error and still have a good copy of the database.

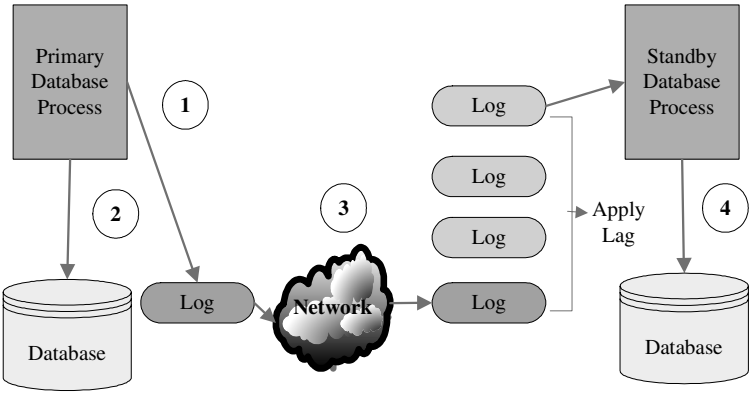

Fig. 6. Using Standby with Apply Lag

The use of a lag comes with a trade-off in safety vs. failover time. For maximum data protection, customers like to configure a long apply lag. This gives them more time to detect and respond to a data failure on the primary before the same data failure also makes it to the standby. But a long apply lag means that there would be more log data on the standby database that are not yet applied. When the standby takes over from a primary database after the primary has failed, the standby needs to apply all the unapplied logs before accepting new transactions. This increases the failover time. Another problem with the apply lag is that queries on the standby database do not see current data.

One way customers have addressed this problem is to have multiple standby databases each configured with a different lag. For example, a customer might have several standby databases that are 5 minutes, 30 minutes, 2 hours, and 1 day behind the primary. When the primary encounters a data failure, they activate the standby with the least lag that doesn't have the same corruption. This is analogous to the way some customers keep multiple snapshots of their data. The difference is that all the standby 'snapshots' are continuously brought up to date (within the constraint of the apply lag).

Oracle Data Guard does not have this trade-off because of the *flashback database* feature. Flashback database [13,14] allows one to rapidly undo recent changes that were made. It does so by undoing all the changes in reverse order. With flashback database enabled, a standby database would immediately apply the changes as they are shipped from the primary database. This way, the standby database is current as of the last change and incurs no delay during a failover. In case of a human error or a

logical corruption on the primary (conditions that previously were handled by the apply lag), the standby would simply flashback the database to a point before the error took place.

4.2 Recovery from Logical Data Failure

End-to-end application checksum validation such as HARD and application-maintained replicas such as the standby database can prevent data failure due to human error and logical corruptions. Nevertheless, there are still situations in which one needs to recover from a logical data failure. This section discusses some of these techniques.

4.2.1 Logical Backup and Recovery

Logical backup and recovery offers finer granularity than physical backup and recovery. For example, database oriented backup and recovery can deal with entire databases, tablespaces, tables, database blocks, and even rows. For backups, control over the granularity means that you can backup different types of data differently. It allows you to perform backups faster and create smaller backup sets. Logical recovery is where application knowledge becomes really important; it can significantly reduce downtime. With Oracle Recovery Manager [15], for example, if only a few blocks in a database are corrupted, you can restore just the damaged blocks and then apply the logs to bring those blocks current instead of restoring and recovering an entire datafile.

Logical recovery also provides much finer control over the point of time to which to recover the data. A database, for example, can use its transaction logs to re-apply the changes made since time of the backup or snapshot. This allows databases to recover to any point in time, given a backup as a starting point. Thus most recoveries would not lose data.

4.2.2 Application-Level Continuous Backups

Oracle has similarly augmented its backup and recovery capabilities with flashback technology. Flashback database is like a continuous backup for the database. To recover the database to a prior point in time, flashback database replays the log in reverse, therefore undoing the operations in reverse order. Unlike conventional recovery, there is no need to restore a backup first. Consequently, flashback is extremely fast when the objective is to undo a mistake or corruption in the recent past.

Flashback database is more efficient than physical continuous backups because flashback database can take advantage of the regular database logs. Like physical restore points, flashback database can also have named points to which you can bring the database back. Because these restore points simply name points in the existing log stream, flashback database restore points do not consume as much resources as conventional split mirror restore points.

Flashback database is a database level continuous backup capability. Some research work has also been done on undoing data failures in mail servers [16].

4.2.3 Precision Data Repair

Application-level data repair can achieve extremely fine-grained data repair. A good example of using application logic to perform fine-grained data repair is an Oracle feature called *flashback query* [13]. Flashback query takes advantage of the Oracle database's multi-version consistency scheme in which the system maintains metadata so that it can reconstruct versions of data in the past. The use of flashback query is best illustrated by an example:

Suppose someone erroneously deleted all the rows corresponding to people who reported to the manager named Jon Smith. That information is now gone from the database.

To get a list of the people that existed at that point in time, one can issue a query similar to the following:

```
SELECT * FROM employee AS OF TIMESTAMP
   TO_TIMESTAMP('2004-12-04 02:45:00',
                'YYYY-MM-DD HH:MI:SS')
   WHERE manager = 'JON.SMITH';
```

This would return a list of employee that reported to Jon Smith at that point in time. Using this information, the data can be reinserted into the database.

Flashback query therefore allows you to find the exact version of the data that you want. Once this is found, it can be reinserted into the table. Since it is expressible as SQL, you can use the full query capabilities to identify the data that was lost and reinsert them.

Along with flashback query, Oracle also supports:

- *Flashback version query* that gives you all the versions of a row between two times and the transactions that modified the row
- *Flashback transaction query* that allows you to see all the changes that were made by a transaction

The combination of the various flashback capabilities makes it practical to surgically repair corrupted data rapidly with minimal data loss. These capabilities cannot be achieved by physical data recovery techniques.

5 Conclusion

This paper makes the argument that logical data protection and recovery techniques are more powerful and offer higher data availability than traditional techniques that work at the physical level. Logical data protection and recovery accomplish this by exploiting application-level knowledge. Physical data protection cannot provide this higher level of data availability. For example, RAID can protect against disk failures but cannot prevent logical corruption or human error. Physical data recovery techniques also have shortcomings in terms of recovery time and data loss.

Fortunately, every physical data protection technique has logical counterpart. Figure 7 summarizes the corresponding physical and logical techniques.

	Physical	Logical
Data Protection	RAID Checksum	HARD (End-to-End Application Checksum)
Data Recovery	Backup/Restore Snapshots Continuous Backup	Backup/Recovery to Arbitrary Points Precision Data Repair Restore Point Flashback Database
Site Disaster	Offsite Backup/ Restore Remote Mirrors	Offsite Backup/Recovery Standby Database

Fig. 7. Physical and Logical Data Protection Techniques

Each of the logical data availability techniques listed is more powerful than its corresponding physical data availability technique. By taking advantage of application knowledge, logical data availability techniques can detect and prevent more errors and offer more and finer-grained data recovery options.

There is still a place for physical data protection techniques. The Oracle database is exceptional in the breadth of logical data protection techniques that it supports. Most applications do not have similar capabilities. These applications must therefore rely on physical data protection techniques. Because physical data protection techniques do not have application knowledge, they can support all applications. Even though it is not as effective as logical data protection, physical data protection is the only choice when there are no logical data protection techniques for the application under discussion.

References

1. Donna Scott, *Continuous Application Availability: Pipe Dream or Reality*, Gartner Data Center 2003, (Las Vegas, NV), page 9, 8-10 December 2003.
2. Tim Wilson, *Ebay retrenches: Devastating outage exposes lack of redundancy, need for simplicity*, InternetWeek.com, June 1999.
   Available at http://www.internetweek.com/lead/lead061799.htm.
3. David A. Patterson, Peter Chen, Garth Gibson, and Randy H. Katz. *Introduction to redundant arrays of inexpensive disks* (RAID). Spring COMPCON'89 (San Francisco, CA), pages 112-17. IEEE, March 1989.
4. W. Wesley Peterson and E.J. Weldon, Jr., *Error-Correcting Codes*, 2nd edition, MIT Press: Cambridge, Mass., 1972.
5. Dave Hitz, James Lau, Michael Malcolm, *File System Design for an NFS File Server Appliance*, Proceedings of the USENIX Winter 1994 Technical Conference, January 1994.
6. Evan Koblentz, *Continuous Backup*, eWeek, June 23, 2003.
7. Symantec's Nortan GoBack 4.0 data sheet.
   Available at: http://www.symantec.com/goback.

8. *Understanding –1018 Errors*. June 2001. Available at: www.microsft.com/technet.
9. *Oracle Hardware Assisted Resilient Data (HARD) Initiative,* 2004, available at http://www.oracle.com/technology/deploy/availability/htdocs/HARD.html
10. *Oracle Data Guard in Oracle Database 10g – Disaster Recovery for the Enterprise*, December 2003, available at: http://www.oracle.com/technology/deploy/availability.
11. *Oracle Data Guard Concepts and Administration 10g Release 1 (10.1)*, Part Number B10823-01, December 2003, Oracle Corporation.
12. *Replication Server 12.6 Features*, 2003, available at: http://www.sybase.com.
13. *Flashback Technology*, 2004, available at:
Available at: http://www.oracle.com/technology/deploy/availability.
14. Oracle Database Concepts 10g Release 1 (10.1), Part Number B10743-01, December 2003, Oracle Corporation.
15. Oracle Database Backup and Recovery Basics 10g Release 1 (10.1), Part Number B10735-01, December 2003, Oracle Corporation.
16. Aaron B. Brown and David A. Patterson, *Undo for operators: Building an Undoable E-mail Store*, Proceedings USENIX Annual Technical Conference, San Antonio, TX, 2003.