CHAPTER 14: REPLICATION

Replication is the process of automatically duplicating and updating data on multiple computers across a network—usually to a geographically dispersed location to fail-safe the data from building loss. The simple description of replication is that it moves data from source computers (those labelled “a” in Figure 14-1.) to destination computers (in this case, computer “b” in Figure 14-1.).

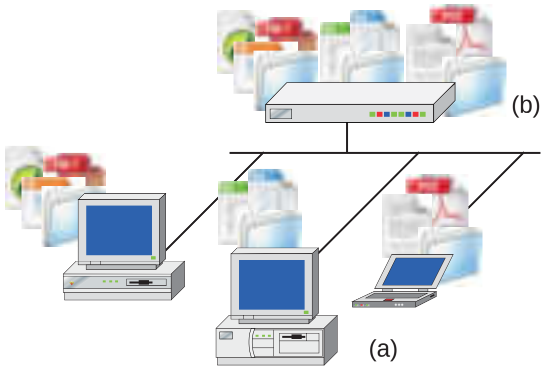


Figure 14-1. Basic file replication

As I said, that’s the short version. It gets a bit more complicated when you implement the replication software, because at that point you have to know about three different variations dealing with the *source of the data*, *how often* that data is moved, and whether the data transfer is *synchronous* or *asynchronous*.

The source data for replication

The replication source data can either be individual files and folders, or whole volumes and partitions. Some products are purely file-focused and aren't used for backing up database volumes; other products are specifically designed with the replication of databases in mind, and therefore focus primarily on moving everything within the volume from point A to point B.

- If the replication process is file-based, the data is examined at the file level and then updated from the source to the destination either in whole or by using the delta block transfer method[1].

  File-based replication systems normally transfer the data from source to destination via any IP-based network. This means that these systems are outstanding for creating mirrored replicas in the organizational network or across the organizational WAN.

- If the replication system is volume-based, the volume is examined block by block, and the replica is updated by that method.

  Volume-based replication systems transfer the information from the source to the destination using either high-speed SCSI or Fibre Channel cabled systems. These systems are outstanding for creating mirrored replicas within server clusters, a campus network, or from one office to the next within a 10-kilometer area (using Fibre Channel).

Either way, replicated files on the destination volume are constantly updated with the changes made to the original files on the source volume so that an exact copy of each and every file targeted is continuously replicated and then available for use on the destination.

Because replication can be either file-based or volume-based, it's an ideal thread-in to network backup practices. It can be used to copy live databases for backup, to centralize data from multiple computer systems to a single computer system for easier backup, and for a host of other purposes.

How often do you want the data moved?

The next thing you need to know about replication is that you can move data in two ways: continuously as it's written, or as a series of point-in-time snapshots.

File-based replication is normally *not* set up to move the data on a continuous basis from the source to the destination, because that just wouldn't make sense. Why spend the computer processes and network bandwidth to move a Microsoft

1. See *Delta block backup* on page 303 for more information on this method.

Word file that gets saved every 10 minutes while the user is working on it? At best, the replication software would have something to do every 10 minutes. Generally, documents just sit there until opened and saved. Therefore, file-based replication is normally set up so that the administrator can designate a regular schedule for when the replication should occur. *By setting up a normally scheduled replication process, an administrator can ensure that the data is consistent between points A and B.*

On the other hand, databases are usually accessed frequently, and therefore replication software that focuses on databases moves that data continuously between points A and B. As we stated in *Open file database backups* on page 343, backing up databases using a point-in-time method causes you to lose a lot of data. Therefore, *allowing the replication process to run continuously will give the administrator peace of mind that the data is not only consistent, but current as well.*

Synchronous or asynchronous transfer

The last bit of replication that you have to come to grips with is how you're going to *move* your data from point A to point B. Moving the data can be done in two ways—synchronously and asynchronously.

The **synchronous** method is simple. When the replication process commences, all the data at the source that you've chosen to move is immediately sent over the network to the destination. If you're moving files during a point-in-time snapshot, this should cause no major problems, as it's handled like any other file transfer through the network. However, if you're trying to continuously move the contents of a database that's being hit hard with a lot of updates, and you're moving large blocks and huge amounts of data at a time, you're now outside the normal bounds of control that a network offers. Your replication system could easily cause your entire network's bandwidth to fill up and become congested during a synchronous transfer.

Therefore, **asynchronous** transfers were devised to curtail those sudden bursts of network traffic that continuous, block-based replication processes create. The asynchronous method of transfer establishes a queue at the replication server in which to hold data while maintaining a steady stream of network traffic between source and destination.

Your choice of using synchronous or asynchronous data transfers really depends on the information you want to replicate, how much change you're implementing to that information, how often that change comes (in lumps or gradually), and how big the network pipes are between source and destination.

The Main Thing

The Main Thing here is that you understand

- **Your business needs,**

  What happens if the data you're replicating isn't the most current? What's the cost of loss of a minute, hour, or day's worth of data loss? Would that cost more than the replication system, or are you trying to propose a replication system that has a price tag higher than the worth of the data you're replicating?

- **Your data transfer needs,**

  Do you need to move data in blocks or files? Do you need to move your data from one office to another office in the same building? Do you need to move your data to another city for it to be safe? You have to ask these questions before you can move on.

- **How often your data changes, and**

  If your data changes relatively often and you're trying to replicate a database, you might need to move to a block-based asynchronous replication process. If your database doesn't change often, or changes gradually, you can probably get away with a synchronous process. If you're replicating files and folders, you'll want to go with synchronous scheduled transfers.

- **The throughput available between your source and destination.**

  No matter which method you use, you need to understand your throughput and latency. Get a good set of network management tools and then manage your network throughput.

Special thanks

Special thanks to John Frassel, Tim Hardy, and Jon Burgoyne over at EverStor for their software, help and support. Again to Connie Chronis at VERITAS, along with Matt Fairbanks who helped a great deal on synchronization and configuration research.

Current, or simply consistent data?

Choosing a replication system is a balancing act: You must weigh the cost of unavailable data against how much you want to spend. And you must grasp another balance—because it'll grasp your wallet one way or the other: the balance of **current** and **consistent data**. Replication helps you with both in the event of your computer's loss by moving that data to a destination that's usually at least a building away from the original source computer. However, the more current you want to keep your data, the more you must spend on connectivity infrastructure between your source server and your destination server. We mention that up front here because it's something that folks don't often think about, yet it could be one of the major cost factors you have to face in planning for replication. Let's start with consistent data and then work our way through to current data.

Consistent data

If the source server dies and the destination server must be made live, your data is **consistent** if the application using it on the destination can be successfully restarted to a usable state with data that's verifiable to a certain point in time—without any corruption (hearkening back to that "write-order fidelity" thing we've mentioned before). While this sounds easy, in practice it can get a tad complicated when you realize how some database transactions work while the replication process is going on.

Let's say that you decided that you're so darn cool you must buy one of the trendy hats at buttlecaps.com.

Consistent operation

1. You sign into the system (you've bought there before) and tell it you want to order a new buttlecap—in this way, you're telling the database manager running the store that you want to modify your purchase records.
2. The database software receives the request in the form of a Data Modification Language (DML) command to perform the update to your record. It then writes this transaction to its log.
3. Once you've entered all of the buttlecaps you want to buy, you hit the **Save** button, telling the database to commit your order to the system as a permanent change. The database enters this as another log transaction. If it's not

busy, it might update to the data table. If buttlecaps are going like hotcakes, though, it maintains your transaction in the computer's RAM until such time as it can write the transaction to disk. However, a write *never* occurs to the table space unless it's first committed to the log.

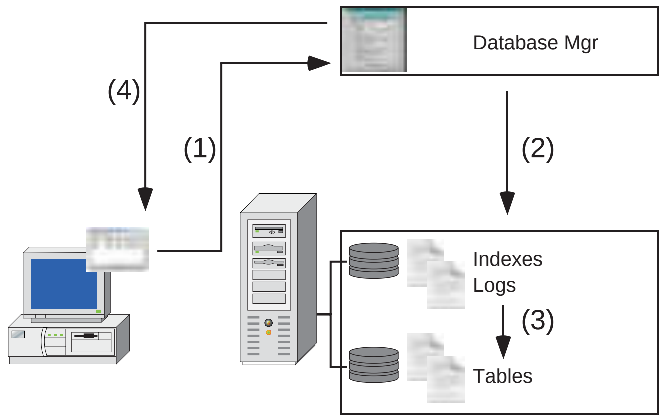


Figure 14-2. Data consistency

4. The database then replies to your computer's application, indicating that the write has been committed and telling it that it's okay to end the transaction.

As you can see from this simple example, the application on your computer is responsible for maintaining the transaction until it receives the commit from the database. That's only the first step in the process.

The database is responsible for maintaining the transaction (sometimes called the **redo**) log. Throughout the normal course of events, the database will checkpoint the log. **Checkpointing** means that all transactions still in memory are written to the disk's tables. After that, the transactions are cleared from the logs and the logs can be archived. In this way, the database maintains consistent data.

Crash and restore operations

Now that we know what a consistent operation looks like, let's examine what happens during a **non-consistent** operation—a fancy term for what happens when the database crashes. For that, we'll put you back in the seat of the user hitting the buttlecaps.com website to order a hat.

1. You've decided to update your buttlecaps order, thus telling the database manager that you want to modify your purchase records.

2. The database software receives the request in the form of a Data Modification Language (DML) command to perform the update to your record. It then writes this transaction to its log.

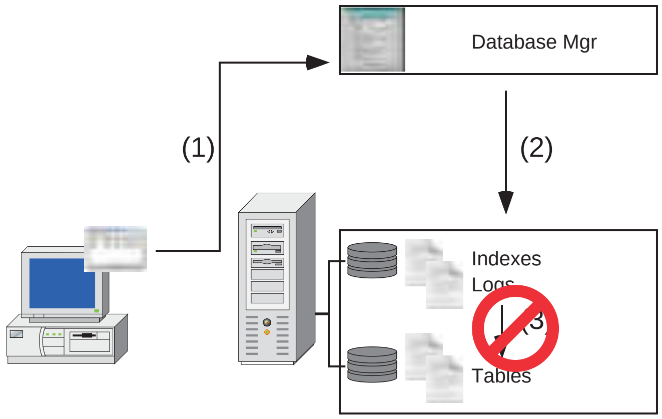


Figure 14-3. Non-consistent operation

3. However, before you can hit the **Save** button and commit your changes, the database server crashes. It won't matter now if you hit the **Save** button as often as you hit the down button in the elevator after a 10-hour workday. The database server isn't there to hear your urgent cries of "I want buttlecaps!" and sooner or later, your application times out.

The database must take several steps to return to a healthy, up-to-date state.

1. First, it must be restarted. This usually occurs after the angst und drang of everyone involved and after the database administrator has calmed down enough to get to the database server administration software and send the restart command.

2. When the database resumes operations, it performs a crash recovery, mounting the data tables in their pre-crash state—the instance of the last database checkpoint. This is called the database's **known state**: some time in the past between the points when the crash and the last checkpoint occurred.

3. Once the database is at a known state, it can then begin to "roll forward" through all of the log transactions and apply the actions to the database. This is the longest portion of time during the recovery process, and can take anywhere from a few seconds (highly doubtful) to hours. All transactions in the

log are committed. Any non-committed transactions (due to the crash) are "undone" in the database.

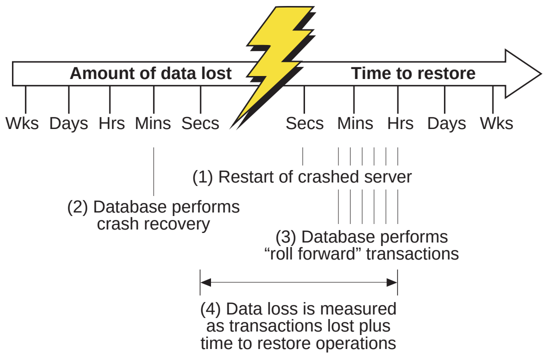

Amount of data lost
Time to restore
Wks Days Hrs Mins Secs
Secs Mins Hrs Days Wks
(1) Restart of crashed server
(2) Database performs crash recovery
(3) Database performs "roll forward" transactions
(4) Data loss is measured as transactions lost plus time to restore operations

Figure 14-4. Database restoration after crash

4. That means that your buttlecap purchase won't be recorded. Had the **Save** button been hit before the crash, and the database received your order and entered the information into the transaction log, the transaction would be marked as committed. The end user would be able to see the record (remember, the server crashed before sending a response to the client app on the end user's machine) only if he or she re-signs in and examines the purchase history. *Data loss is measured as transactions lost plus time to restore operations.*

Of course, all of this depends on the database's ability to restart.

What happens if the database crashed due to a fire in the building that destroyed the computer? Or if the building fell down? Or, if you lived in my old neighborhood, where it's likely that somebody just came in and took the computer? Then there wouldn't be any computer to restart the database *with*.

Let's look at the restoration process rebuilt from a tape or disk backup of the data.

1. First, if your computer is gone, you must replace and reconfigure it. You can do this in less than an hour if you swapped in an existing computer and reloaded the software and setups from an imaged disk system.

2. Your next step: Layer on top of your image the data that was lost in your database. This could take anywhere from less than an hour to hours.

3. You're left with whatever data you had during your last off-site tape backup—up to a week's worth of data loss. That isn't good.

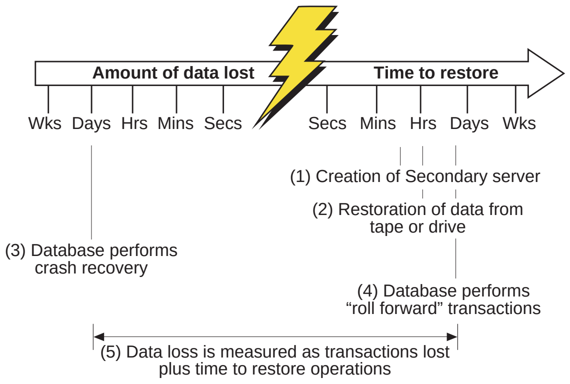

Amount of data lost
Time to restore
Wks Days Hrs Mins Secs
Secs Mins Hrs Days Wks
(1) Creation of Secondary server
(2) Restoration of data from tape or drive
(3) Database performs crash recovery
(4) Database performs "roll forward" transactions
(5) Data loss is measured as transactions lost plus time to restore operations

Figure 14-5. Database restoration from tape

4. Once your database is up and running, it must still go through the "roll forward" process (assuming that you've backed up the transaction logs, as well).
5. Your data and productivity loss spans a much greater time interval than it would have in a mere computer crash.

Which, of course, is why you've wisely chosen to replicate your data to an offsite computer. Let's walk through what happens during an asynchronous replication, and how it recovers in such a situation.

Asynchronous replication

In an **asynchronous replication** system, data is written first to the primary source and then to a source cache, which queues the data for transmission to the destination as bandwidth allows.The queue acts as a buffer between the source and the destination, providing a much kinder replication environment for your network than synchronous replication does. When the writing application creates surges in the update rate, the queue grows. As the data is received by the secondary, the queue shrinks. This prevents the writing application from being bogged down waiting for data to transmit from the primary to the secondary.

However, in the event of an emergency, asynchronous replication is more susceptible to variances in data between the source and the destination. If the source dies before a transmission has been sent, or packets are delivered out of order, data is lost. If the source dies with a large buffer, the information therein won't be transmitted, causing a discrepancy between the information in the source and the information in the destination.

To handle this additional complexity, replication management software must be added as a layer between the database manager and the volumes being written to. Also, a replication journal that works with the replication management software must be created, and installed, with the replication manager in the receiving, secondary computer. On the receiving end, there should be no live database manager trying to access the replicated database—that would gum up the works. The diagram below (Figure 14-6.) shows the basic setup of the primary and secondary database, as well as the additional steps necessary for a transaction to take place. Let's walk through it.

Consistent operation

1. You've decided to update your buttlecaps order; thus, you're telling the database manager that you want to modify your purchase records.

2. The database software receives the request in the form of a Data Modification Language (DML) command to perform the update to your record. Instead of writing the information directly to its transaction log, it passes the information to the replication manager, which writes the information to the primary server's replication journal.

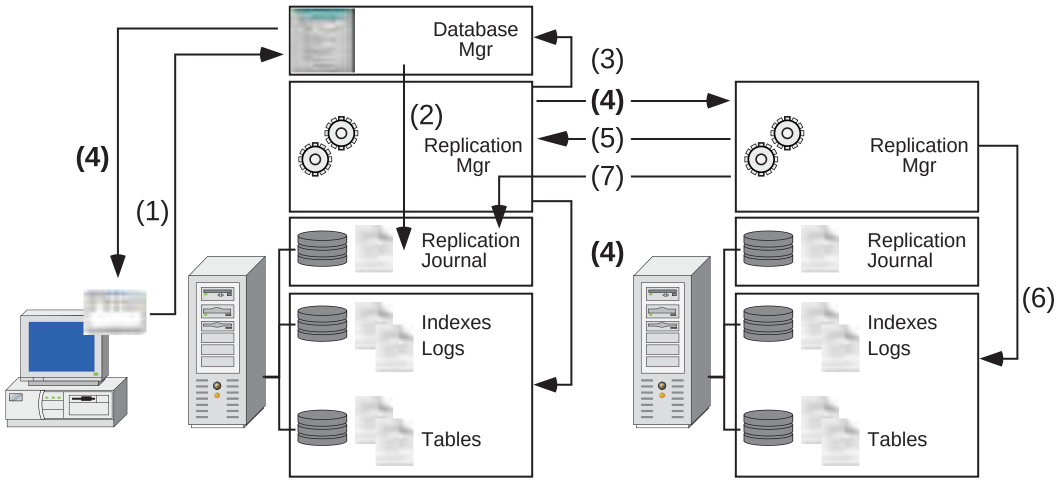


Figure 14-6. Asynchronous write process

3. Once you've hit the **Save** button, instead of the database manager writing to its transaction log, the Replication Manager again steps up to the plate, does the work, and tells the database application that the write is complete.

4. At this point, several things happen simultaneously. The database manager replies to the client that the transaction is complete. The replication manager writes the data to the primary volumes, and at the same time, adds the information to the outgoing queue to be sent asynchronously to the secondary host.

   At this point, more data might have been sent than the connectivity between the two hosts can handle. In this case, the queue begins to grow, creating a backlog on the primary server. We'll cover just how much backlog can grow in another section—for now, understand that only so much data can go through a pipe at one time. This isn't Wiley Coyote who turns on the spigot for the hose and watches a bulge go through—alas, real life is not nearly as colorful as a cartoon.

5. Upon receipt of the data, the secondary host sends a network acknowledgment to the primary host, stating that the data is present and in memory, ready to be written to disk.

6. The secondary host then writes the data to its local disks and sends an acknowledgment to the primary host.

7. Once the primary host receives the acknowledgment from the secondary host that the data has been written to local disk, it marks the write as complete in the replication journal.

In a database environment (whether that database is a simple MySQL system, a complex Oracle system, or the structured and incredibly complex file storage engine of a bookkeeping system), updates are made to various elements in a fixed-sequence methodology that can be spread over multiple directories, or even multiple volumes. If this data gets out of sequence, the database or bookkeeping system reject it. Any replication system that works in these environments *must* consistently safeguard the in-sequence writing of this data—VERITAS calls this **write-order fidelity**. This state can be achieved either through software (such as VERITAS' product) or through using asynchronous transfer mode (ATM) wide area networking, or a combination of both.

Crash and restore operations

Let's look at the process again—but this time, there's a hitch: The primary server dies in the middle of a transaction.

1. Let's say that you started the process as before, but you're expanding your TQ (trendiness quotient), and want to buy a different buttlecap for every day of the week.

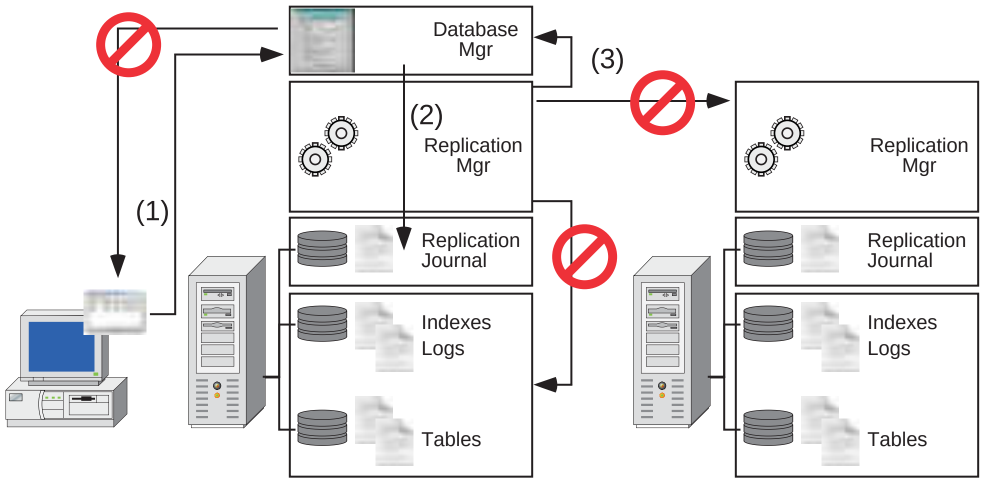


Figure 14-7. Death of a server

2. The replication manager starts to do its thing by logging the request into the replication journal.

3. It then passes the information back to the database manager that everything is hunky-dory. But before the database manager can send a reply to the client, or the data can be written to the primary storage volumes, or the data can be queued for sending to the secondary volumes, the unthinkable occurs: *The computer dies*. A massive typhoon has hit the Ogunquit office that housed the primary database, and the database is swept out to sea. Months later, some happy inhabitant of an uncharted island will have a dandy new coconut-smasher—but for now, let's return to civilization as we know it.

Since it has received no acknowledgment, the client computer will eventually either hang, or the end user will give up. You've now lost your primary server, along with any data in the primary server's queue. That's the bad news.

The good news? Restoring your system to productivity is *much* faster, because part of the secondary's job is to monitor the primary server's heartbeat (like all of us monitoring Cheney's to ensure that George doesn't take charge). If the secondary notices that the primary has missed a consecutive number of heartbeats and has lost contact, it immediately begins the recovery process, which is much like the recovery process of a server that has crashed. To the secondary, there is no loss of hardware—it doesn't need any imaging or data restoration from a backup operation. Here are the steps it takes, and the amount of your loss.

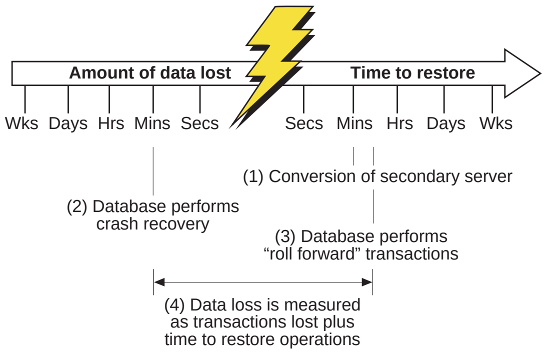

Amount of data lost
Time to restore
Wks Days Hrs Mins Secs
Secs Mins Hrs Days Wks
(1) Conversion of secondary server
(2) Database performs crash recovery
(3) Database performs "roll forward" transactions
(4) Data loss is measured as transactions lost plus time to restore operations

Figure 14-8. Secondary recovery process

1. The secondary senses the death of the primary and converts into active mode.
2. Even though this version of the database hasn't crashed per se, it still performs a full recovery operation from the last checkpoint.
3. It immediately begins to roll forward through the transaction logs.
4. The data loss consists of all of the information in the transfer queue on the primary computer (and which therefore hadn't yet been sent to the secondary), those transactions that were still live and not yet committed to the database, and the recovery time it took to switch the secondary from standby through live and the roll-forward operations. In essence, this could be anything from minutes to hours, but it sure beats having to restore the data from a backup and *then* begin whatever roll-forward process it can.

The only scenario that involves less downtime and less data loss is accomplished through synchronous replication. Caveat emptor here: Unless you have one heck of pipe from the primary to the secondary server, your system won't transfer a lot of data. I don't know what type of pipe it takes to run a synchronous replication for our servers over the WAN, but I do know that it would be expensive.

Current data

The obvious benefit of synchronous versus asynchronous replication is the reduction of data loss—but that benefit comes at a cost. Because of the complexity of the network connection, network management, and monetary considerations, a move from asynchronous to synchronous replication is ultimately a business decision.

In a perfect world, the secondary and primary hosts have almost identical information. If data loss is measured in minutes for the fastest possible asynchronous replications, it's measured in seconds for the synchronous system. The data lost in an asynchronous replication consists of those transactions that were pending, but not committed, as well as all of the data in the replication queue. The data lost in a synchronous replication consists solely of those transactions that were pending, but not committed. For this amount of data to be moved from the primary to the secondary, you need a very fast and wide connection. Let's explore synchronous replication in more detail.

Synchronous replication

In a previous chapter, we defined how the network protocol stack works, with its multiple layers speaking to each other (*The basics—a layered approach* on page 128). In a **synchronous replication** environment, the writing of data from the source to the destination is done at a very low layer in the protocol stack, so that the write update to the destination is acknowledged by the source before the operation is actually completed at the Application layer on the source. This must happen very fast, or it slows down the source's performance waiting for the data to be written. The synchronous method ensures that even if the source fails, the maximum amount of data has already been transferred to its destination.

Before you plan on deploying synchronous replication, you'll need to understand it and *y*our network's capabilities in the throughput and latency categories. In researching this book, we found out that roughly 32 percent of organizations recently interviewed by *InformationWeek* said that they had the network infrastructure to support high volumes of instantaneous data delivery[2]. For most of the world, synchronous replication is most effective in application environments that have normal LAN characteristics and low update rates within the data structure. It can also be deployed effectively in write-intensive applications, as long as a high-bandwidth, low-latency network connection is pervasive.

Current operations

1. You've decided to update your buttlecaps order—you're now telling the database manager that you want to modify your purchase records.
2. The database software receives the request in the form of a Data Modification Language (DML) command to perform the update to your record. Instead of

2. "How to build networks with zip—these technologies can help keep your network reliable and responsive," by R. Gareiss, *InformationWeek* (2002).

writing the information directly to its transaction log, it passes the information to the replication manager, which then writes the information to the primary server's replication journal (VERITAS calls this the Storage Replicator Log). *So* far, you've followed the exact same steps as you did in the asynchronous mode.

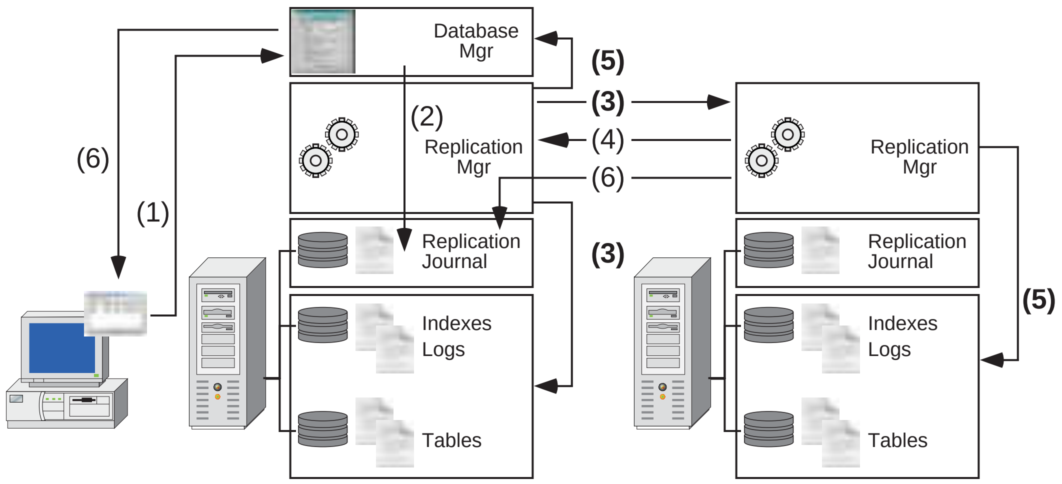


Figure 14-9. Synchronous replication process

3. In synchronous mode, this step consists of two simultaneous steps: The replication manager writes to the secondary host, and while waiting for the network acknowledgment from the secondary, it writes the information to the primary data volumes.

4. Once the secondary receives the write information and processes it in its kernel memory, even before it writes any data to its resident volumes, it returns a network acknowledgment to the primary.

5. In this step, more simultaneous processes occur. The replication manager on the secondary writes the data to its volumes. As soon as the primary receives the network acknowledgment from the secondary, it informs the database application that the write is complete.

6. As the write has taken place on the secondary's volumes, the replication manager marks it as complete in the replication journal. The database manager notifies the originating client that the transaction is complete.

The huge difference between synchronous and asynchronous modes? In synchronous mode, the primary writes data to its own drives while it sends the data to the secondary, and further, that write won't be acknowledged to the database manager until both primary and secondary writes are complete.

If a primary server crashes (or the building crashes down upon it) midway through the write phase (step 3, above) so that the write phase doesn't happen, the only real data loss consists of those writes that were about to happen, and any pending requests that might have been opened but were never saved or committed to the system. Let's go through the steps of a worst-case crash and restore operation:

Crash and restore operations

1. You started the process as usual: Deciding to boost your TQ, you want to buy a different buttlecap for every day of the week.
2. The replication manager starts to do its thing by logging the request into the replication journal.

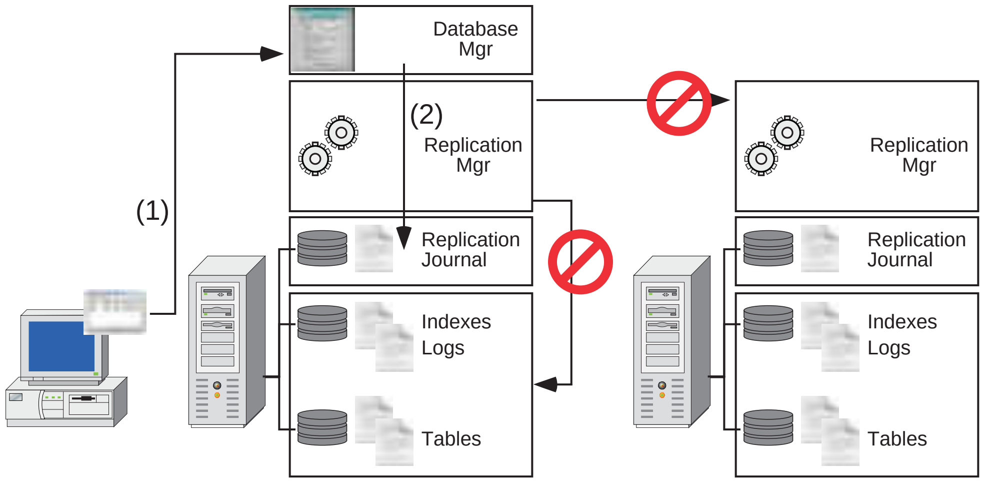


Figure 14-10. Synchronous database crash

3. Before or during the process of writing to the primary's volumes and sending the information to the secondary host —*the computer dies*. A massive tsunami has hit the Pismo Beach office where the primary database is located, and the building that housed the database collapses, washing its contents out to sea. On yet another uncharted island... well, you know all about arcane uses of flotsam and jetsam. However, the secondary database was housed across a fi-

ber link two blocks away, and neither the building nor secondary host sustained any damage.

Because synchronous systems are also usually tied in with clustered server management systems, you'll know within seconds that the primary has failed and the system will automatically jump into fail-over mode, bringing the secondary host to live status.

1. The secondary senses the primary's demise and transfers into active mode.
2. Even though this version of the database hasn't crashed per se, it still performs a full recovery operation from the last checkpoint. However, since the data is about as live as live can get, only seconds' worth of data isn't written to the secondary host's volumes.
3. It then immediately begins to roll forward through the transaction logs. Again, since synchronous operations are nearly simultaneous, there isn't much of a roll-forward log.
4. The data loss consists of all the information that was being written at the time of loss, those transactions that were still live and not yet committed to the database, and the recovery time it took to switch the secondary from standby through live and the roll-forward operations. In essence, this can be anything from seconds to minutes—and that's about as good as it's *ever* going to get.

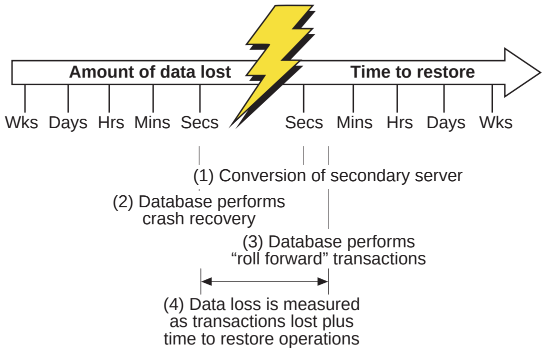

Amount of data lost
Time to restore
Wks
Days
Hrs
Mins
Secs
Secs
Mins
Hrs
Days
Wks
(1) Conversion of secondary server
(2) Database performs crash recovery
(3) Database performs "roll forward" transactions
(4) Data loss is measured as transactions lost plus time to restore operations

Figure 14-11. Recovery operations

The Tools

The tools that we review here are all *replication-specific*. We also discuss (albeit in an oblique way) Retrospect's ability to duplicate information from point A to point B. However, since this application is not a replication-specific engine, we don't cover it directly in this section. It is, however, covered throughout the book, and duplication-specific information is covered in *Document and directory duplication* on page 277.

Replicator

EverStor's Replicator is a server-based asynchronous replication engine that runs on Unix, Linux, Mac OS X, and Windows-based computers or Network Attached Storage devices. We think it runs best on NAS boxes or an Apple Xserve. This is one heck of a program, and we love it—it's easy to use and presents the user with a web interface for restoration. Replicator uses a three-step process to move files (it doesn't work with databases) from the client computer to its storage space. Replicator sits on top of the operating system.

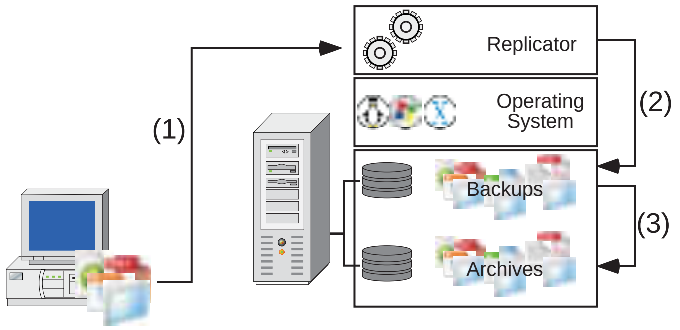


Figure 14-12. Replicator

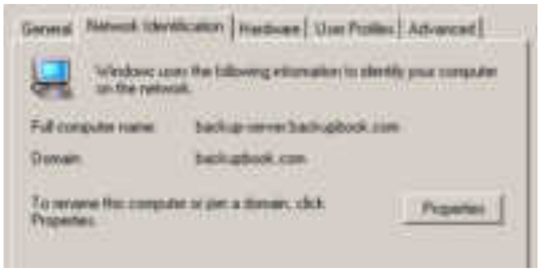
1. Replicator pulls files from the clients at intervals set by the administrator. It can pull files through FTP or SMB.

2. Once it has the files, Replicator stores each of the client file sets in predetermined areas of the hard drive with permissions for access to those areas that are tied to each of the workstations.
3. Because Replicator is more than a duplication engine, it allows *N* number of versions of each file to be stored in its archive directory. So, if a file is being added during step 2, before it overwrites the present version, it archives that file according to the rules set by the administrator.

Machines

Machine is Replicator's term for an individual client computer. Each client machine managed under Replicator must have a separate machine definition. As system administrator, you can manage each of these machines individually or delegate responsibility to others.

Remember that part about naming devices that we talked about in the network corruption chapter (see *LANsurveyor* on page 153)? It definitely comes into play here, as machine names cannot be duplicated in the Replicator system. On Windows devices, you can find the device's name by right-clicking on **My Computer**, then choosing **Properties** > **Network Identification**.



Windows uses the following information to identify your computer on the network.

Full computer name: backup-server.backupbook.com

Domain: backupbook.com

To rename this computer or join a domain, click Properties.

Figure 14-13. Name of Windows 2000 device

For Linux and Unix, the machine's name can be determined by typing the **Hostname** command at the command prompt in the terminal.

For Macintosh OS X users, select **System Preferences…** then click **Sharing** in the window to bring up the computer's sharing information.

Once you have the basic information in hand, you're ready to set up the device's replication parameters. Since Replicator provides a web-based management front end, you can set it up from any location and on any computer that has a standard web browser. It's pretty simple: Just add the device's name or IP address, the device's directory name for replication and archiving (usually the same name as the user device), how many iterations of the same file are allowed (they call this the *max number of archives*), and a log level for this user (if the user is having problems, you'll want to log it).

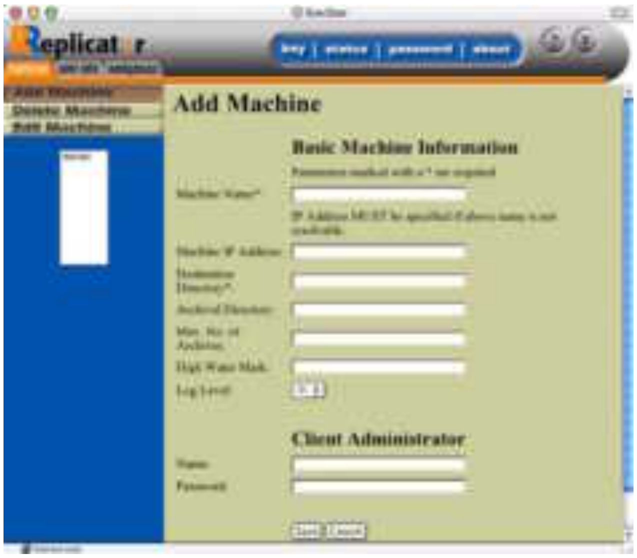


Figure 14-14. Machine information

It takes longer to figure out the device's name than it does to set up the device in the system.

Save Sets

Save Sets are defined for each machine, determining a set of files to be replicated, and how and when the replication is to be performed. The Save Set is the definition of the data to be replicated on each client machine. This definition includes the directory or directories to be replicated, those to be excluded (if any), the frequency of replication, and the maximum permissible levels of archival.

Since the Replicator product has no client software, the software uses SMB or FTP to access the local computer's source directory. FTP is the preferred method of setting up a share-point on a local Unix/Linux computer, and SMB, of course, is for Windows users. Mac OS X users can set up FTP share-points or SMB share-points—however, setting up an FTP share-point is much easier.

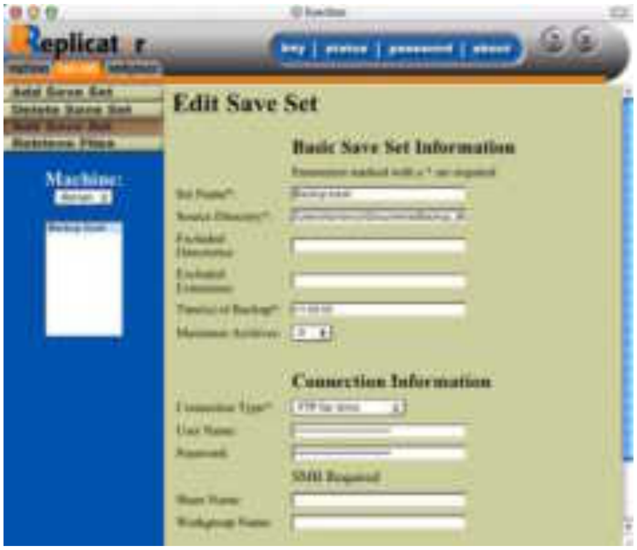


Figure 14-15. Save Set information

Workgroups

Workgroups consist of a group of machines managed by the workgroup administrator, and are used to delegate responsibility. The Replicator system administrator may designate workgroup administrators as responsible for groups of client machines. In a large organization, this can be a huge plus, but since this is a logistical issue and has nothing to do with functionality, we won't cover it here.

Replicator is a fine product for moving files from point A to point B. Because it doesn't use any software on the client source computer, it must have a very specific address (or a resolvable domain name within the network) to find the source volume. This makes it great for replicating general user directories, websites, and any other static file-based volume from the source to the backup destination.

It doesn't work, however, with any open files—so using it to replicate a mail server or a database server is out of the question. Any file that is not an open file can be quickly and easily replicated to the destination server where Replicator is running. And since it supports all major operating systems, we think it's great.

VERITAS Volume Replicator

Based on the VERITAS Volume Manager, VERITAS Volume Replicator (VVR) replicates volumes of data in real-time (either synchronous or asynchronous) to remote locations over IP networks. Why the plural emphasis (locations)? VVR can replicate one volume group to 32 secondary replication sites simultaneously (if you have enough bandwidth or your volume doesn't have a heavy change load). The ability to replicate entire volumes in this manner is an extremely robust storage-independent disaster recovery tactic. Unlike Replicator, VVR sits beneath the operating system, intercepting hard drive calls and routing them appropriately.

VVR follows roughly the same set of steps in both synchronous or asynchronous modes of operation.

1. As writes come in from the database application, VVR intercepts the call.

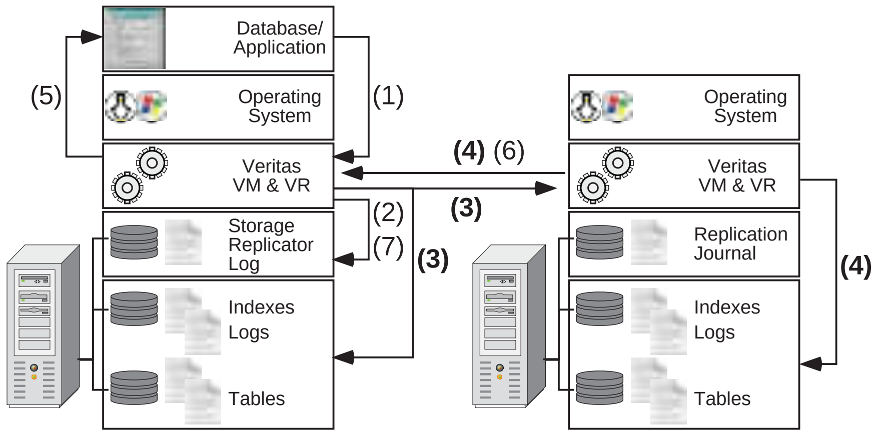


Figure 14-16. VERITAS Volume Replicator

2. VVR writes the change information to its Storage Replicator Log, so that if a crash occurs before the rest of the procedures is finished, the information is contained in its replication log and can be used for restoration purposes.
3. Once the write to the Storage Replicator log is complete, VVR sends the write to the secondary host (or hosts). In asynchronous mode, this data is written to a queue that then manages the sending of the data. In synchronous mode, the data is sent directly. At the same time, VVR writes the data to the primary computer's volumes.

4. On the secondary hosts, VVR receives the incoming write, processes it (putting the information into its kernel memory), and sends a network acknowledgment to the primary host.

5. In synchronous mode, once the primary receives the network acknowledgment from all of the secondary hosts that the data has arrived, VVR sends an acknowledgment of the write's completion to the application.

6. Once the secondaries have actually finished writing the data to their local volumes, they send a write acknowledgment to the primary.

7. Once the primary receives write acknowledgments from all of the secondaries, VVR marks the write as complete in the Storage Replicator Log.

Replicating databases as volume groups

Remember when we were talking about write-order fidelity (*Consistent operation* on page 501), how databases process the writing of information in a certain order, and how that order must be maintained in order to recover from a crash? Good—you were paying attention. Go grab a gold star and a Krispy Kreme. Now comes the part where you get to put all your diligence to use by planning out a VVR replication setup the correct way.

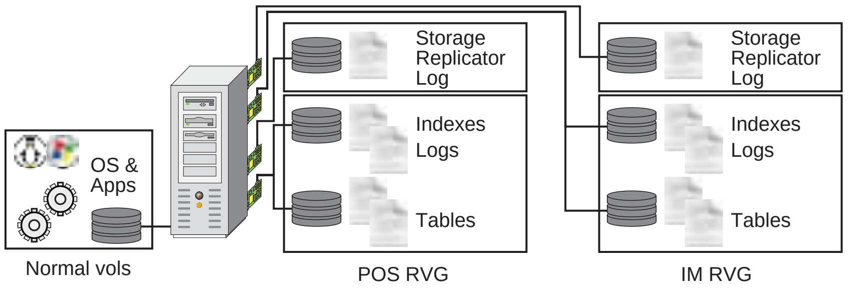


Figure 14-17. Replication Volume Groups

Let's say that you have a single large computer running both a POS database for your 30 stores as well as a second Inventory Management (IM) database for the same 30 stores. Each of these databases writes independently to its respective volumes and database transaction logs. To maintain write-order fidelity for each of the independent databases, you must set up independent **Replicated Volume Groups** (**RVGs**) for each database. An RVG is a group of volumes within a VVM disk group (Remember those? We talked about them in *Dynamic volumes* on page 478)—and all the related volumes are part of the same disk group.

Here are some tips for RVG setup:

- Separate each database into its own RVG, as we've done in Figure 14-17.
- For extra safety, make sure that you mirror all your SRL volumes, as well as all data volumes.
- Make sure that you name the volumes on all of your secondary hosts the same as those on the primary host's volumes. In a disaster situation, your secondary will become your primary and you'll want volume-naming fidelity if and when you have to map your drives over.
- Dedicated separate physical disks (with separate physical SCSI or FC controllers) to your SRL volumes, separating them from your data volumes. Ensure that your data volumes have separate physical disks and controllers as well.

Setting up the asynchronous RLINK

VERITAS calls the network connection between the primary and each secondary host a **Replication Link** (**RLINK**). The RLINK's attributes specify the replication parameters, whether synchronous or asynchronous, for the corresponding secondary host. If the amount of data sent by the primary is greater than the RLINK can handle, one of two things happens (other than your users start complaining that the network is pokey). In the synchronous replication mode, the primary server must wait until the writes are complete on the secondary server—and this could greatly impact the performance of your primary server. In asynchronous mode, the SRL queue grows until the change rate is less than the pipe size, and the queue can empty as the overflow gets sent to the secondary. Therefore, you must set up the data transmission link between the primary and each of the secondaries appropriately.

Below (see Figure 14-18. on page 521) is a graph showing the relationship between…

- The data pipe (ours shows a 100 Mbps network pipe);
- Normal network traffic (which doesn't amount to much here);
- The rate of data change taking place on the primary (that must therefore be sent across the data pipe to the secondary host); and

- The SRL that has to hold the excess data until it can be sent from the primary to the secondary.

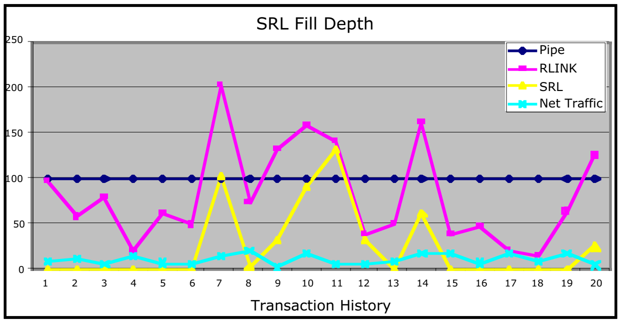


Figure 14-18. SRL fill depth as it relates to data flowing through an RLINK

The SRL begins to fill up with a backlog anytime that the amount of data that must be sent to the secondary is greater than the amount of data that can flow in the pipe at that time. *If the primary goes down with data in the SRL, that data is lost.* As you can clearly see in Figure 14-18. , wherein the fill depth of the SRL at points nine through 11 spike as the data rate mushrooms, a 100 Mbps pipe for this RLINK between the primary and secondary would be a bit small, even though there isn't much *normal* network traffic (think what would happen if the network traffic spiked, as well as the RLINK traffic?). In the diagram below (Figure 14-19.), we show an RLINK with even more traffic than before, but now the same traffic between the host and primary is running on a gigabit network instead of a 100 Mbps network. The SRL fill depth is never above zero because the pipe between the primary and secondary can easily handle the load.

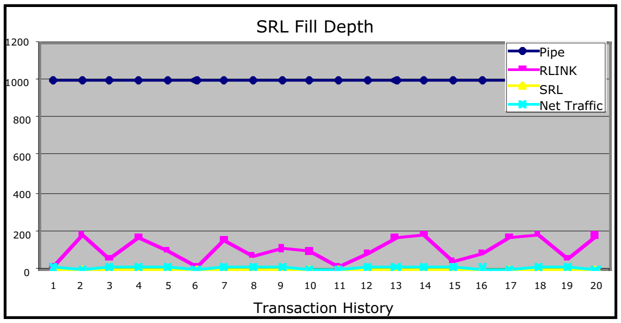


Figure 14-19. The RLINK moved to a gigabit network

Our suggestion? When you establish an RLINK between a primary and secondary host, build an RLINK-only network. By adding a secondary NIC card to both the primary and secondary host(s) and then routing that data directly to its own LAN or WAN, you eliminate any problems with LAN traffic snarls, give yourself a much cleaner path, and can easily monitor this network (with simple tools like CyberGauge from Neon Software, which we've mentioned throughout this book) to ensure that the data rates aren't soaring too high.

For heavy-use databases on a LAN, we suggest that you utilize a gigabit-switched network. For heavy use between a primary host and a secondary host over a WAN, we suggest that you create dedicated point-to-point connections (such as T1s, T3s, or better) between each device.

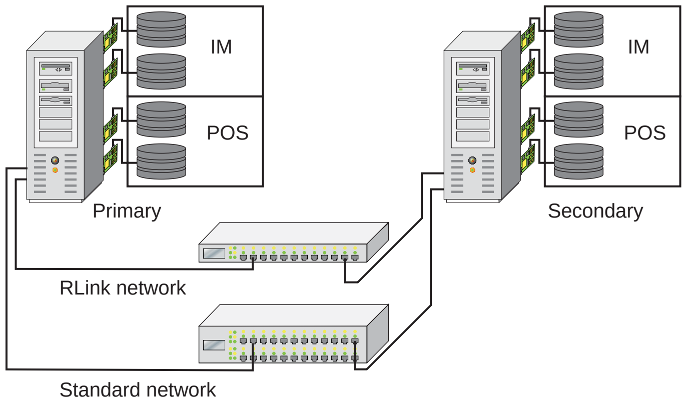


Figure 14-20. An RLINK network

Remember, this is key data, and you're creating a mirror through VVR to protect it. Don't foul up your protection scheme by cheesing out on the network connection—that's the least expensive part of the whole operation.

VERITAS Storage Replicator

VERITAS Storage Replicator is a file replication product that provides real-time replication for workgroup-level systems. It's ideal for consolidating the backup process to a centralized location within the Windows platform, or replicating databases from one point to another. Storage Replicator keeps an up-to-the-minute copy of specified volumes, directories, or files to allow for immediate recovery, by mirroring the source-file system writes to a destination server as they occur, operating with full fidelity on open files. Storage Replicator can be used to maintain real-time copies of databases stored in a safe location in a different part of the campus or in a remote location.

When replicating, Storage Replicator sends updates from the primary node on which the application is running, secondary or remote node. Replication is unidirectional (updates on the primary are sent to the secondaries, but access to the data at the secondary nodes is read-only while replication continues). If the data at the primary is destroyed by a disaster, the copy of the data at the secondary can be made write-accessible, and applications that were running on the primary can be brought up on the secondary. The secondary can then be used as the new primary for the writing applications.

Storage Replicator has three main components.

1. **Replication Management Server** software that holds the configuration data for the replication systems, controlling the beginning and ending of the replication process. This software drives the process and is the repository for all of the logs, alerts, and histories.
2. The **Replication Service Agent** is client software that is installed on every computer that is designated either a source or a destination. The Replication Service Agent must be installed on the same computer as the Management Server *if* that computer acts as either a host or a destination. Each Replication Service Agent is dedicated to a single Replication Management Server (to avoid conflicting jobs).
3. The **Console** is an application that provides a front end for information about the replication configuration and replication processes. It's great that VERITAS has separated the front end from the engine so that the administrator can remotely manage a replication server, or can have multiple copies of the console so that certain tasks can be delegated to subordinates who've installed the Console on their computers.

Storage Replicator ties all of this together into a "replication neighborhood" (with one server acting as the Remote Management Server for the whole neighborhood) and uses job management terminology to describe its process. A replication "job" defines the source and destination volumes, the data being replicated, the interval and scheduled duration of the replication, and a few specific settings that are germane to each replication process.

Storage Replicator works by placing itself in the computer's kernel stack just above the operating system in order to intercept the 64 k writes to the disk so that it can not only write the information to the primary host, but also to the replica destination. Because this is an asynchronous system, Storage Replicator simultaneously passes the incoming writes off to the operating system (2) and also adds the data to its replicator journal for processing and sending to the replication destinations (3). This works exactly the same whether files are being written to the drive or database writes are being performed.

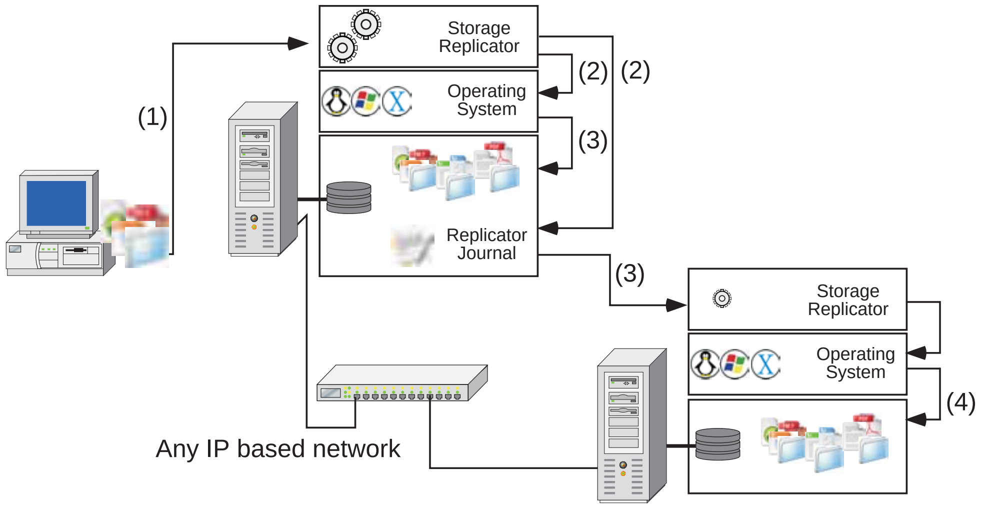


Figure 14-21. Storage Replicator process

Because Storage Replicator can replicate data over any IP network (that supports the throughput and latency necessary to accomplish the job), the primary source computer can be in one location and the replica destination can be elsewhere, such as the company hot-site, or another office or building. And because Storage Replicator works as the operating system is *writing to the disk*, live databases do not need to be shut down to be moved from point A to point B.

When a job is started, the source volume is not considered protected until each and every specified file has been replicated onto the destination and then checked to ensure that it is identical to the source file. We'll cover this in more depth below, but briefly, a synchronization phase is followed by a dynamic update phase. Storage Replicator first synchronizes a complete set of files between the source and the destination. Once the list of files has been synchronized, it then goes through the list to update any files to be replicated. For explanation purposes here, we'll walk through the process of creating a replicant database structure instead of a file structure, since the database structure is more volatile and specific.

Synchronization phase

When a replication starts, it synchronizes copies of files between the primary source and the destination replica, so that an exact list of all files targeted exists in both places. Therefore, the first step is to build an exact copy of the file structure on the destination computer. Any files that don't exist on the replica are copied from the primary to the replica. Any files that do exist are examined, and if changes are found, a new copy is transferred from the source to the destination (if the file is smaller than 1 MB), or the delta changes to the file are updated from the primary to the replica.

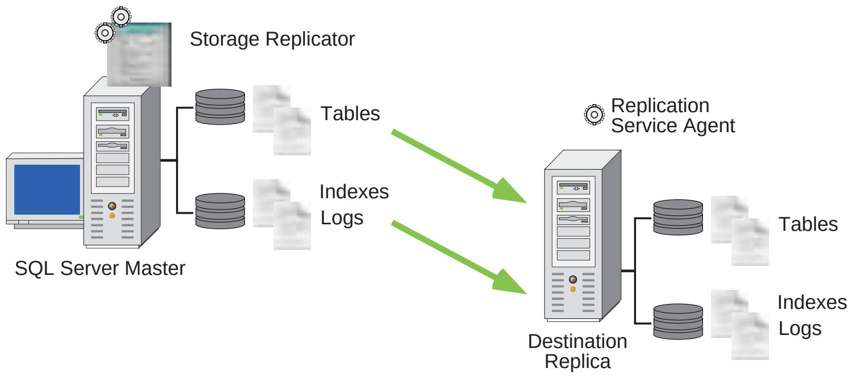


Figure 14-22. File synchronization phase

The only problem with this process is that since the database can be live with changes being made to it, as files are replicated from the primary source to the replicated destination, they can be out of date once they've hit the destination server because changes have been made to the primary copy of the file. At this moment, the dynamic phase comes into play.

Dynamic phase

The **dynamic phase** of replication tracks changes to the files as they are occurring on the source (remember above, where the Storage Replicator software intercepts the operating system calls to the drive? That's how it knows). Once synchronization has been completed, any changes to the primary files are then sent to the destination, updating the copies of those files on the replica.

To maintain write-order fidelity, the dynamic phase tracks the changes on the primary in the order they occurred, updating files on the destination file in the same order, thus guaranteeing the database's transactional consistency[3]. Once all of the files have been synchronized and then updated, the replication process is complete.

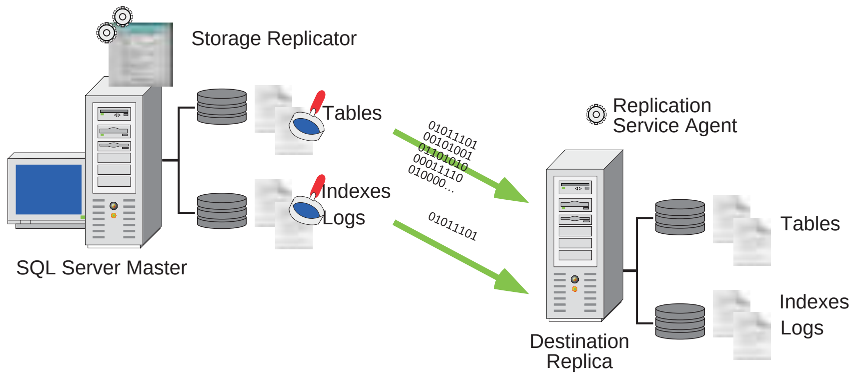


Figure 14-23. Dynamic phase

The ongoing process

At this point, the administrator has two choices. An ongoing, continuous replication can occur from here on out, or the replication process can be stopped and then restarted on a scheduled basis.

If the replication process is stopped and then restarted on a scheduled basis, further synchronization will take place, followed by the dynamic phase.

If the replication process is allowed to continue on an ongoing basis, only the dynamic phase will have to run, as the data will be moved to the replica destination as it is written to the primary source. During dynamic replication, Storage Replicator will continuously send changes on the source volume over the network

3. Storage Replicator is a *file replicator*, and as such doesn't understand SQL database tables, logs, etc. and therefore isn't used to move and reintegrate them. Instead, it replicates files ensuring write-order fidelity so that the data is a perfect match, and if necessary, can be run from the destination.

to the destination volume. In Storage Replicator, this happens asynchronously, using as little CPU processes on the source as possible. To do this, Storage Replicator's drivers record all the changes to each document and place them in journals.

Journaling

Because Storage Replicator uses an asynchronous replication process, the change can't be sent immediately from the source to the destination. Rather, a copy of the changed portion of the file is placed in a **journal**, transmitted over the network and written to a journal on the destination, allowing for temporary network bottlenecks. An inbound journal on the destination device that ensures all synchronization data (the creation of new files) is written first and then followed up with dynamic data.

Journal files are self-managing files that grow and shrink with the amount of data to be transmitted and then integrated into the destination. These journal entries are transferred across the network as quickly as the network allows (using as much bandwidth as allocated within the console of the software). Once arrived, each journal file will typically hold around 3 MB of data. Large job transfers can take up to several individual journals.

In planning for this additional space on both source and destination, take the amount of files being transferred. Add 10 percent for the additional journal size. Journal files (on both source and destination) won't be deleted until after the journal has been marked as read by the software. Then multiply that combined number by a fudge-factor of 15–20 percent more for the total size you're going to need.

If your network is up to speed (pardon the pun), you probably won't see a journal entry file on either source or destination. However, a total failure of the network connection between source and destination, or network corruption causing near-zero TCP Window problems on either one (See our *Low window size* on page 165 for more info), expands the source journal. High CPU contention or usage on the destination expands the journal on the troubled device and delays replication processes, causing potential data loss if the primary fails during that period.

Planning the Replication Link

Using the VERITAS RLINK estimating tool, we created a single RLINK between a primary and secondary host to replicate a 4.5 Gb database. The database changes by about 3 percent a day, and the 100 MB LAN is used to transport the data (with 10 percent traffic on the LAN).

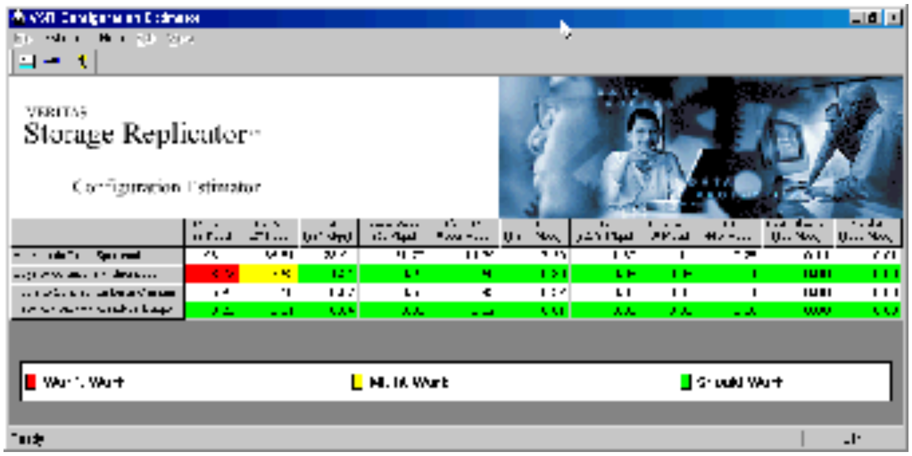


Figure 14-24. VERITAS Configuration Estimator

The VERITAS Configuration Estimator reveals that our computer will complete the initial synchronization in about six to seven minutes, and synchronize the delta changes even faster. Synchronizing over our T1 WAN connection takes almost seven and a half hours for the initial synchronization and about another half hour for the delta changes. Quite an improvement, I'd say!

USES FOR IP-BASED REPLICATION

You can employ several tactics to optimize IP-based replication. We've listed the applications that work in each specific instance in Table 14-1. through Table 14-3. to help you select the right software for the task at hand.

Live DB to standby DB replication

To maximize your database's uptime, perform a replication of your database to another SQL server running different SQL databases (and ignoring this one), or with the MS SQL software preinstalled but not running (standby mode). How does this maximize uptime? If you replicate the data in a SQL server to SQL server mode, you can simply launch the standby server in case the primary fails completely, or you can map the standby server's drive to the primary's drive if the primary's drive fails.

In both low- and high-volume scenarios, you can use either Dantz Retrospect *or* VERITAS Storage Manager. However, you *can't* use EverStor Replicator, as it doesn't replicate open files.

Live DB to Standby DB	Retrospect	EverStor Replicator	VERITAS Storage Mgr	Storage Replicator
Low Volume Database	✔		✔	✔
High Volume Database			✔	✔

Table 14-1. Live DB to Standby-DB Replication

When replicating the database data from the primary SQL server to the standby, you can choose to replicate either individual databases or all the databases on the server. It depends on the importance of the information, and the amount of bandwidth between the two servers. When planning your SQL to SQL mapping, plan as if you were going to map the entire database set, just in case you need to do so later on. Therefore, make sure that your source directory and replication directory are the same for your SQL data.

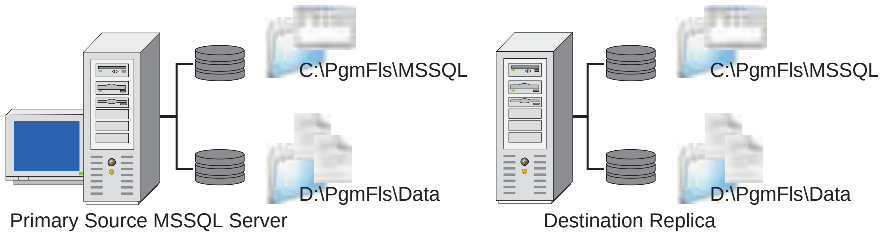


Figure 14-25. Mapping MSSQL and data directory information

Then, if your data volume on your primary source SQL server dies for some reason, you can map your replica drive over to the primary, and the database engine won't know the difference.

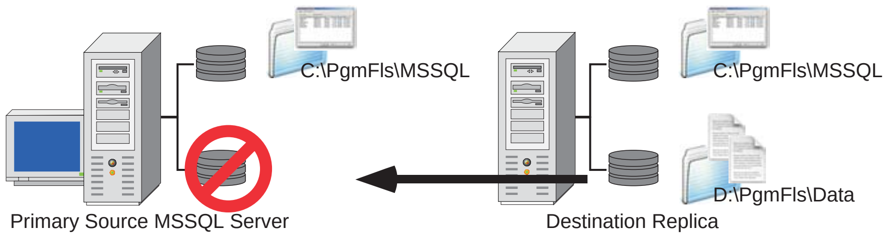


Figure 14-26. Remapping a downed volume

Or, in case of catastrophic failure, you can launch MSSQL on the replica server and have that act as the primary.

Remember when we talked about not having the budget to create clustered servers and the fact that you might want to plan on doubling up your systems in the event of a failure (see *What if you can't afford a clustered server or a SAN?* on page 494)? Well, here, we show you how to implement such a system. Let's say that you can't afford a cluster, but also can't afford to have your database system down, so you set up your replication engine to replicate the database in your back-office server over to your normal database server. You then set up your database server to replicate its database over to your back-office server.

When setting up your servers to replicate to each other, you'll probably want to have separate partitions for each of your data volumes and your boot/applications volume. By setting up separate drive volumes and letters for each partition, when you replicate the information from the source to the destination, you won't overwrite anything. Also, you can *pre*-set up each local database application to access either data source. By doing this, if the remote computer's data source goes down, you can ensure a smooth transition for bringing it live on the surviving computer system.

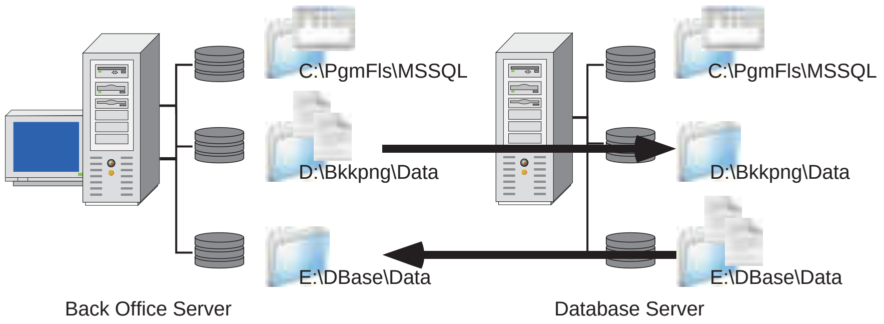


Figure 14-27. Double-duty server operations

You can also create a round-robin scenario using this method with two or three different database servers in different locations. Let's say that you have a home office, as well as two remote offices in Ogunquit, Maine, and Pismo Beach, California.

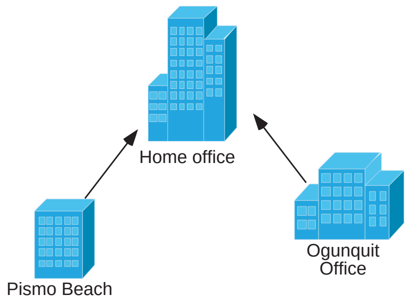


Figure 14-28. Round robin replication

You could replicate your home office to Ogunquit, Ogunquit to Pismo Beach, and then Pismo Beach back to your home office. This method allows very fast recovery of all of your servers, and ensures that *some* office will be up and running if a catastrophe strikes.

Live DB to file server replication

Storage Replicator can also be used to replicate live databases from their primary source over to a network file storage system (like a NAS box), so that they can be backed up using normal backup processes. This is considered a many-to-one replication, because many databases can be consolidated onto a single file server for backup centralization. In low-volume scenarios, you can use Dantz Retrospect's duplicate function and open file manager or VERITAS' Storage Manager. However, you can't use EverStor's Replicator, as it doesn't replicate open files.

By moving the databases off-host to a centralized location, you can make your backup processes much simpler because you won't have to worry about any other open file manager systems running on the SQL database server.

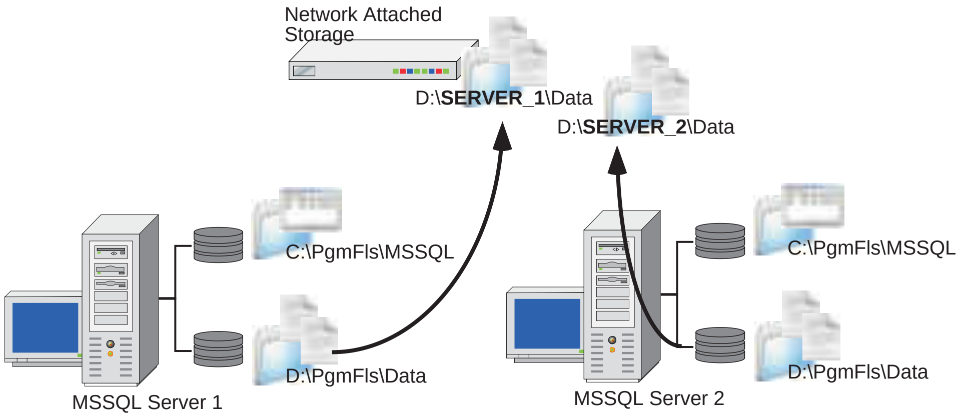


Figure 14-29. Replication to a file server

In this scenario, when using both Retrospect and Storage Manager, if you use a regular file server as your destination, you can run the server software on that file server on another device, such as the backup server.

Live DB to File Server	Dantz Retrospect	EverStor Replicator	VERITAS Storage Mgr	Storage Replicator
Server software can be run from NAS box			✔	✔
Client software can be run on NAS box	✔		✔	✔

Table 14-2. Live DB to file server replication

However, if you use a Network Attached Storage (NAS) box as the file repository, Retrospect isn't the answer, because it requires a front end (that the NAS box really doesn't give you). Since VERITAS' console doesn't have to be on the server, you can run Storage Manager's Replication Server Manager software on the NAS box and the console elsewhere. Since EverStor's software would choke on a live database without an open file manager, we've excluded it from the list of candidates here.

File and directory consolidation

Replication can also consolidate information from many places to one, as shown in Figure 14-30. This allows the backup admin to run centralized backups of servers only, knowing that all pertinent data resides on the servers and that the workstations can be retrieved from an image, if necessary.

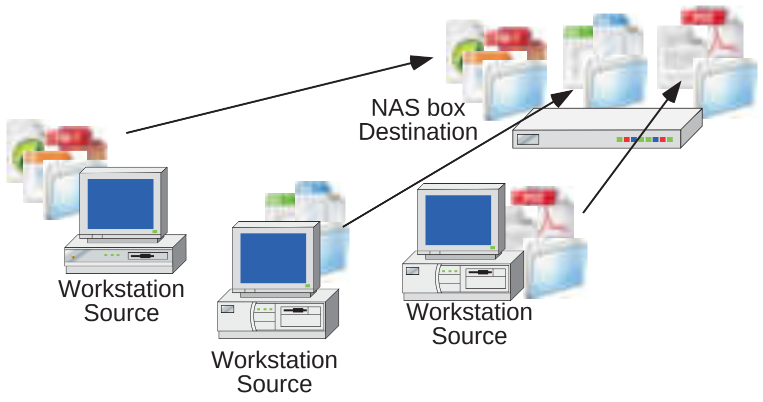


Figure 14-30. File and directory consolidation

Many corporations are geographically dispersed, but still need a centralized backup solution. In this case, how can you maintain local backup and archiving needs as well as a centralized backup solution without plunging into Administrative Inferno? Because this is an IP-based file replication scenario, you aren't limited to moving files and directories of information from single computers to a central server on your LAN—you can use this type of replication to move whole data sets from remote offices back to your home office through the use of the same tool. That means that you can, in essence:

- Consolidate reports on select computers from multiple offices to a single office;
- Consolidate application or file servers from remote offices to a central office

for the purpose of centralized backup and recovery; and

- Create a tiered system wherein each remote office consolidates files onto a single server, and that server replicates itself up to the home office, where all of the data is then stored onto a backup tape system for history and long-term archiving and recovery.

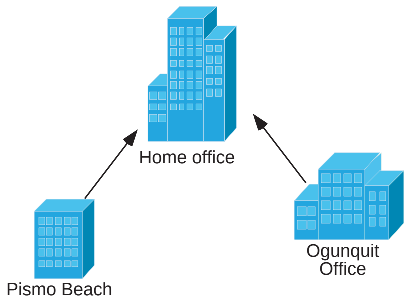


Figure 14-31. WAN consolidation

The only application limited in this regard is the VERITAS solution, as it has only Windows 2000 client software and won't work in a tiered environment. However, it does work for replicating one layer of servers up to the next layer. Each of the other tools allows all three operating systems to replicate their data from one place to the next.

File Consolidation	Dantz Retrospect	EverStor Replicator	VERITAS Storage Mgr	Storage Replicator
Windows Client	✔	✔	✔	✔
Mac Client	✔	✔		
Unix Client	✔	✔		
Allows multiple tier usage	✔	✔		

Table 14-3. File consolidation