opentext™ | Cybersecurity

Carbonite Replication Theory of Operations

Technical Overview

Purpose

This document provides a high-level overview of the basic theory of operation for Carbonite® Availability, Carbonite® Recover and Carbonite® Migrate replication engine on a Windows® server. It details the flow of data from the originating server (source) through the network to the destination server (target). This document requires a fundamental knowledge of basic Carbonite Availability, Recover and/or Migrate operations. This guide focuses on our engine, which is used by different Carbonite Availability, Recover and Migrate workflows. It is a very flexible engine and allows for selective replication of individual files and folders or for the whole system state (OS, data and applications). Separate documentation is available if required.

Carbonite Availability was formally branded as DoubleTake Availability, and Carbonite Migrate as DoubleTake Move. Originally only a Windows solution, a similar set of solutions is now available for Linux environments as well.

Please bear in mind that Carbonite Availability and Carbonite Migrate share the same replication engine, and several of the same wizards and in many ways work the same. The primary difference between Carbonite Availability and Carbonite Migrate is that Carbonite Availability is designed to protect your data in advance of an unforeseen incident to ensure minimal data loss. On the other hand, Carbonite Migrate is used to migrate your data to a platform in a managed way with zero data loss and near zero downtime.

Origination of Data

For the purposes of this discussion, the term “data” will refer to any digital information that can be visualized by a server as a series of one or more input/output (I/O) operations. Data is generated by the Windows system itself, in the form of updates to binaries and day-to-day changes in various system files. Data is more frequently generated by applications that run on the server. This data may be non-transaction dependent (such as read and write operations on flat files) or transactional-dependent (such as read and write operations to database systems). In either case, the data will be translated into a stream of I/O operations that will be processed by the Windows file system and committed to disk. Data may be created by end-users, applications, or any other system that can read and write information to the server. It may also be destined for local disk resources, virtual disks (VMDK/VHDX), remote (SAN/DAS) resources, or any combination of the above.

Carbonite Replication File System Filter Driver

Carbonite Replication uses a file system filter driver to integrate itself within the I/O stack on a server. This filter driver allows the Carbonite Replication process to see each I/O operation that flows through the stack to the file system. Any read/write operation performed by any application or by the Windows operating system will be visible to the Carbonite Replication driver and therefore, may be subject to replication depending on how workflows are configured.

The Carbonite Replication filter driver is a kernel-mode driver. This means that it exists within Windows kernel memory – a subset of system memory reserved for high-level operations that cannot be accessed directly by general applications or end-users. This will permit the Carbonite Replication filter driver to act with minimal impact on the server itself, while still ensuring that each I/O operation is able to be examined. Since filters exist within the server’s OS kernel, several safety features have been implemented to ensure they do not adversely impact the system in the event of an error.

- First, the filter itself only examines data and allows for a copy of required I/O operations to be moved out of the data stream. Carbonite Replication’s filter driver is non-blocking, so it will not halt data flow in the event of an error. Instead, should an abnormal condition be created on the server, the Carbonite Replication filter will allow data to pass through un-scanned, producing an error event to alert administrators, but not stopping applications and end-users from accessing data.
- Secondly, even when operating under normal conditions, the Carbonite Replication filter allows other applications (such as anti-virus scanners) to operate without hindrance, so that non-Carbonite Replication operations may continue as they did before installation of Carbonite Availability or Carbonite Migrate.

Selecting the Data

Our intuitive wizard simplifies the technical aspects making it easy to choose the data that needs replicating. Advanced users are empowered to choose which data goes where at a detailed level. But the engine also automatically excludes transient data, so less advanced users do not need to ‘deselect’ options like the recycle bin or the swap file.

The replication set is the fundamental unit of data protection for Carbonite Replication on a server. It defines which areas of a file system will be protected or migrated, and which will be ignored by the filter driver during normal operations.These definitions are made using either the management console GUI, or via one of the workflow wizards – such as the File and Folder Protection/Migration, Full Server Protection/Migration or Full Server to ESX Protection/Migration templates.

Replication sets are defined as a series of rules which indicate to Carbonite Replication which files within the server should be replicated. The replication set may contain a group of files, a directory, a group of directories, volume or group of volumes.

In addition to the basic definitions, replication sets can also be modified by the user to include and exclude files by wildcard (*) definition. Administrators may define extremely specific replication sets to protect only what a specific application would require for resumption of services or may instead protect or migrate a much more general set of data – up to and including an entire server. The replication set information is stored in a configuration file called DblTake.db.

The Carbonite Replication service will use the replication set rules to determine which replication operations from the driver should be sent to the target. If the Carbonite Replication service sees replication activity from a file that is not within the replication set rules, then the service will notify the driver to not send any more information regarding that file. The filter will still see change activity for that file, but it will not send any further replication data to the service. Because of this, it is important to note that any changes to the replication set rules, while the connection is established, will require that the connection be disconnected and reconnected to ensure that the filter driver is reset with the updated rules. It should also be noted that no replication data is being passed from the filter driver until a Carbonite Replication connection is established with a target.

Initialization of a Connection

Once the replication set is defined, it can be connected to a target device via the Carbonite Replication Console. This will cause a series of events to occur based on the options selected during connection. They are:

1. **Carbonite Replication ensures** that the target system is responding that the proper version of Carbonite Availability or Carbonite Migrate is installed on the target and that the Carbonite Replication systems on that target are ready to receive information.
2. **The connection is established** using the replication set rules previously defined and the target path that was provided in the connection information. If the target path does not exist on the target, it will be created.
3. **Once the connection is established**, a replication start command will be sent to the filter driver which will cause the driver to start sending change data. At this time, a mirror is started. The mirror will enumerate the files within the replication set and will read the data from disk and send it to the target. The file structure will be created on the target underneath the target path provided by the connection information. The administrator may choose either a full mirror (over-writing any data on target) or differences only (by file-system block checksum) when creating the connection, and the chosen form of mirror operation will initiate on connection. While the mirror is occurring, any changes to the data within a replication set will be replicated to the target.
4. **After all the mirror data** has been received by the target, the mirror process will update the attributes of the target directories to ensure that the attributes and last modify times are correct. The mirror operation will then complete, and the connection statistics will report that the mirror is idle.
5. **Replication will continue** as long as the connection is established.

Replication Mechanics

The replication process for Carbonite Migrate, Carbonite Availability and Carbonite Recover is based on the same replication technology which is described in the following section.

Note that replication is only active during protection or migration, and that replication of data is a continuous process that only ends when:

1. The source and target loose communication to the point where queuing is no longer possible.
2. A failover event occurs.
3. The administrator directs Carbonite Replication to stop protection.

Outside of those events, any new data written to any area of the file system within the replication set will be replicated and written to the target.

Once the connection is established and protection is enabled, the Carbonite replication engine performs a series of logical steps to ensure that data synchronization between the target and the source servers occurs.

Our engine runs a unique algorithm that creates two dynamic processes that run in parallel. On a macro level, there is an initial mirror that runs and makes sure everything is synchronized. Then on a micro level, there is a replication that intercepts disk changes and gets real-time and byte level data.

In a sense, the engine runs through a sequential process, but will double back if needed . This is all done to wrap the engine in a series of fail safes with checks and balances at every stage in the process.

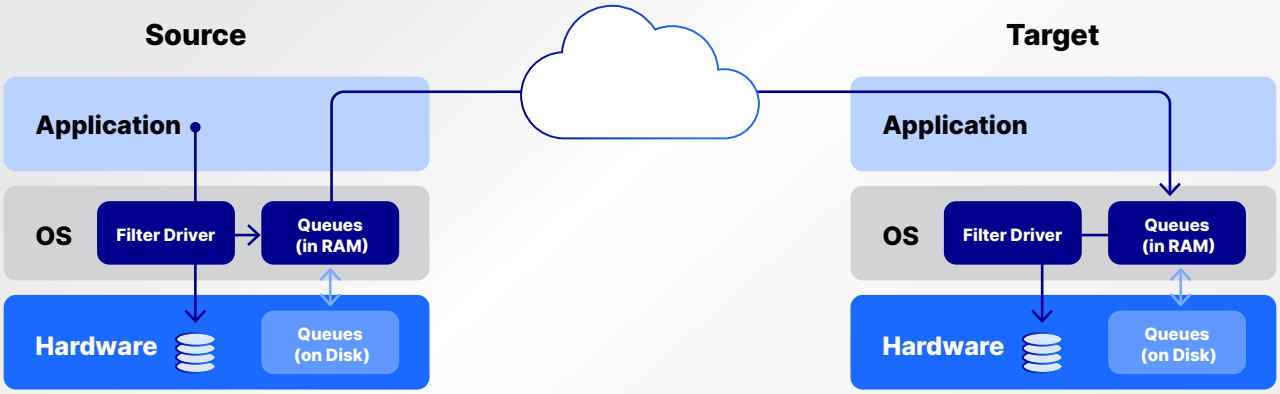
Here are the steps that the engine takes to ensure a comprehensive replication:

- The Carbonite replication engine verifies that the service is running on the source server. While the service is running, all disk I/O operations are passed to the Carbonite replication filter driver.
- The Carbonite file system filter driver intercepts requests targeted at a file system. By intercepting the request before it reaches its intended target, a minifilter can extend or replace functionality provided by the original target of the request. I/O requests which pass through a filter are required to pass back through the filter after being executed by the disk sub-system.
- After the I/O request is executed, the response is evaluated by the filter driver against two attributes:

- Replication set: Was the I/O request for a file or folder location defined as within scope by the replication set rules.
- I/O Type: Was the I/O request a write operation. This includes any write operation (write, update, delete) that will result in a change to the file system.

- If the resulting evaluation is true in both cases, then the filter driver will split / copy the write request response. Note: If the I/O is a read operation (meaning that it will not change the data), then the operation is passed through with no copy made.
- The response is then returned to the requesting application for acknowledgement and the copy is directed to the Carbonite replication service together with an assigned transaction ID.
- The redirected request is then queued for transmission over the TCP networking stack to the target server as a write request. Replication queues are held in RAM but will buffer to disk if there is system load or network latency delaying transmission.
- Once the write request is received by the target server, it is delivered as a disk I/O request where it is again passed through Carbonite replication service on the target host.
- Once committed to disk, the filter driver notifies the Carbonite replication service, and the sync operation is marked as complete.

Data Transmission



The Carbonite Replication file operations are temporarily stored within the Send queue on the source system. As always, the order of the operations is maintained within the queue and is consistent with the order in which data was written to disk. The send queue acts as a buffer to the network and if the source cannot send data to the target as fast as it is being written to disk, then the send queue will store Carbonite Replication operations that need to be transmitted. Data will technically always flow through the queuing systems to allow for instantaneous queuing when an emergency bottleneck emerges, but barring such slow-downs or disconnections, file operations remain in the queues for a brief period before being transmitted. The Send queue will hand off file operations to the native Windows TCP/ IP systems for packetization and transmission across the IP network between source and target. Before transmission, a copy of the operation is placed in the Acknowledgement queue on the source, so that if there are any errors within the transmission process, the operation can be re-transmitted.

The Windows IP systems transmit the operations as a series of TCP/IP datagrams over administrator-specified ports where the target system then receives the information via the target's TCP/IP Windows stack. The CarboniteReplication service running on the target will take the data from the network, where sub-systems examine the I/O and perform several integrity checks. Included among these checks are tests to ensure that the I/O has been received in its entirety, in the correct Windows I/O transactional order, and without errors. If any test fails, the target will request a re-transmission of any necessary operations from the Acknowledgement queue of the source, and the necessary packages are retransmitted and re-checked for integrity. Since all these tests and checks are performed on the copy of the I/O taken from the acknowledgement queue, the source is not required to re-read live data.

Once the file operation has passed all integrity checks, an acknowledgement packet is sent back to the source, at which point the source will remove the now successfully transmitted file operation from the Acknowledgement queue, freeing up queuing space. On the target, the validated file operations are passed to the Receive queue, where it will either temporarily be held (if there are any disk bottlenecks, etc.) or sent to the file system to be written to disk.

Since each file operation is written in the exact same order as it was written on the source, transactional integrity for any application writing through the file system will be preserved explicitly. This is true across multiple files or within a single file. This is how Carbonite Replication can safely protect or migrate Microsoft SQL, Oracle and other forms of transaction-dependent data sets.

Example Scenario - Microsoft SQL 2019

Using Microsoft SQL as an example of an application whose data is protected, we can follow the path of a SQL transaction through the Carbonite Replication system. First, using the Carbonite Replication Console, a replication set is defined using a wizard. For SQL, the replication set would protect all database (.mdf) and log files (.ldf) files. Additional files and folders may be added to the replication set if desired. After the replication set is defined, a connection is established between the source and a target. An initial mirror is initiated, and replication systems begin protecting the data. Once the mirror has completed, then data on the source and target is in sync.

As the live SQL database is modified by users or applications, it generates changes to its data files, Carbonite Replication will replicate that data in the same order that it was written on the source. For transactional recovery, SQL creates a log write, followed by series of read and write operations in the databases, followed by another log write. Though this is an oversimplification of the entire technical process, it does provide a useful outline of the steps taken.

Carbonite Replication's file system filter driver can see the entire series of I/O operations and replicate the write or change data operations during that series. We capture those disk writes, and all write operations are passed to the Carbonite Replication Service, in the same order they were generated. While the Carbonite Replication Service is transmitting the captured write operations to the destination server, the original disk write instructions proceed without any modification on the source. In other words, capturing and transmitting the changes has no impact on the normal local disk writes that are being made on the source server. Carbonite Replication will ensure that each I/O operation has been successfully committed to disk and prepare each file operation for transmission and place it in the Send queue. The operation will then be sent to the target as quickly as possible.

Carbonite Replication receives the file operations on the target and writes them in the exact order as they were written on the source. In this case, Carbonite Replication would be writing first to the log file, then a series of writes to the database and finally another write to the log. Of course, this is happening in as close to real-time as the link and systems will permit, so it is possible that the tail-end of this operation was finishing on the source as the first writes were begun on the target. Also keep in mind that since we write each I/O operation in the same write order as SQL made that write on the source, a failure of the source in the middle of transmission would have no greater impact on SQLthen a roll-back to the last committed SQL checkpoint when a failover occurs. This is a normal operation for SQL and is supported by the write-order integrity of the Carbonite Replication system.

Conclusion

Through this process of write-order intact byte-level I/O replication, Carbonite Replication can safely protect or migrate any Windows or Linux based system. This includes Exchange, SQL, SharePoint and File servers, and extends to any other applications running on the server, including the operating system itself. Further details of the different workflows, which will allow you to protect or migrate complete system state or modernize SQL and SAP are available. They all use this same Carbonite Replication engine.

Please speak to your Carbonite representative for further details.