Filesystem Scans
Copyright © 2020 Datadobi, All rights reserved.

Datadobi believes the information in this publication is accurate as of its publication date.

The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” DATADOBI MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any Datadobi software described in this publication requires an applicable software license. DobiMiner® is a trademark registered in the US Patent and Trademark Office. All other product and company names are trademarks or registered trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them. Linux is the registered trademark of Linus Torvalds; Unix is the registered trademark of Novell; Java Standard Edition (SE) and Java Development Kit (JDK) are registered trademarks of Oracle Corporation; Microsoft and Windows are registered trademarks of Microsoft Corporation; Dell EMC, Isilon, VNX (1/2), and Unity are registered trademarks of the Dell Technologies Inc.; Hitachi is the registered trademark of Hitachi Vantara; Hewlett Packard Enterprise and StoreEasy are registered trademarks of Hewlett Packard Enterprise Development LP; and NetApp is the registered trademarks of NetApp, Inc.
INTRODUCTION

In this technical brief, we will discuss the ways in which differences between systems in a migration or replication data copy are determined. In any data copy, the speed at which differences can be determined before the start of the copy is of critical importance. We will show why scanning the filesystems is the most reliable way to detect changes, and why legacy tools are simply too limited when compared to the modern capabilities of the DobiMiner Suite™. We’ll also discuss why file change notification application programming interfaces (APIs) are not a solution to the challenges faced during data copy.

Note that we are not addressing the methods by which in-family data synchronization is accomplished. Instead, we address the challenges posed when dealing with data copies across network-attached storage (NAS) platforms from different vendors. When synchronizing data across disparate platforms, in-family tools cannot be used.

MIGRATION/REPLICATION PHASES

A cross-platform NAS migration or replication can be broken into a number of phases, with the first phase being to seed content from the source system into the destination. At Datadobi, we refer to the seeding of data as the first copy operation.

Once the first copy is complete, the data mobility engine settles into the steady state. This is an iterative phase whereby automated incremental copy operations are executed in order to keep the source and destination systems synchronized. When performing data migrations or even planned cutovers to a replica system, the speed of incremental copy iterations is especially important, because the cutover event involves the last incremental copy before users and applications are redirected to the new destination.
The overall process of executing incremental copies can be broken into three key repeating phases (Figure 1):

1. A scan of both the source and destination systems to capture the current state of all filesystem objects. The filesystem scan is the foundation for all other phases, as an accurate reading of the source and destination filesystem maps must be known.

2. Differential (Diff) calculations are run on the scan results to determine any filesystem object changes that may have occurred on the source system since the last incremental copy. This determines the delta copy operations required to synchronize the two platforms.

3. Delta copy operations are then executed to synchronize the two platforms. New files that have been added, files that have changed, metadata updates such as permission changes, file deletions, etc., are all propagated to the destination system.

Figure 1 – Steady state/incremental copy phases
CHANGE NOTIFICATION APIS – HOW THEY WORK

Why scan the filesystems given that notifications are generated when changes are made to files and/or directories? Changes made on NTFS filesystem objects, for example, can be detected using the FileSystemWatcher class as part of the .NET framework. On Linux®/Unix® systems, the inotify interface and/or incron daemon can detect individual file/directory changes. The Java® SE Java Development Kit (JDK)® allows a Java application to intercept filesystem events.

Given that today’s NAS devices can hold billions of files, the question still remains: Isn’t it fundamentally inefficient to scan filesystems on both the source and destination to identify when changes have been made? In the context of executing cross-platform NAS migrations, the answer is an emphatic, “no.” The data collected from a filesystem scan provides the most accurate results for determining the delta operations to synchronize the two systems.

To demonstrate this, we will examine the three notification types mentioned above – these are not the only options, but represent commonly used methods for monitoring changes. In all cases, scanning the filesystem to detect changes is not required. Rather, an application merely needs to accept notifications, and keep a list of the filesystem changes. This list (data update, metadata update, new file, deletion, etc.) is then used to generate copy operations based on the accumulated changes.

1. **NTFS – the FileSystemWatcher class.** This .NET class can be instantiated in an application that monitors filesystem change notifications and raises events when a directory, or file in a directory, changes. The application must run on the Microsoft® Windows® file server that locally mounts the shared file systems.

2. **Linux – the inotify interface.** Linux’s inotify is an inode-based filesystem notification mechanism that essentially works through “watches,” or monitors. The watches specify an event mask such as opens, reads, writes, creates, metadata changes, etc. Inotify is based on system calls as opposed to library functions, so it relies on the Linux kernel being properly configured to send and receive notifications.

3. **Java.** The Java SE JDK provides the WatchService API, which allows a Java application to register types of events for which it will receive notifications. For example, these events can be file creation, file modification, file deletion, etc. When the service detects relevant events per the registration list, notifications are sent back to the application.
The .NET FileSystemWatcher class and the Linux inotify watcher both sound reasonable, but there’s a catch. These capabilities are available on the server platforms that own the underlying filesystems – i.e., they work for local filesystem change notifications. While the FileSystemWatcher class can be used to detect file/directory changes on a Windows server, it’s unreliable for Server Message Block (SMB) shares provided by a NAS device. Similarly, inotify can only be used on a Linux system as it requires support from the Linux kernel to work, and can’t detect changes made on filesystems not mounted locally.

NAS vendors don’t typically implement inotify on their systems, nor does the network file system (NFS) protocol (v3, which predates inotify) provide support for filesystem change events. A host with an NFSv3 export mounted from a remote system can’t monitor the kernel space.

As with both the .NET FileSystemWatcher class and the Linux inotify interface, an application implementing the Java WatchService API must run on either a Windows or Linux server, and is dependent on that server’s kernel to dispatch notification events. A Windows/Linux server running a Java application incorporating the Java WatchService API can’t mount a filesystem from a remote server or NAS device and still receive these notifications.

NAS DEVICE CHANGE NOTIFICATIONS – HOW THEY WORK

What about leveraging NAS device capabilities for cross-platform migrations or data replication operations? This would be an option if: a) all platforms had the ability to provide notifications, and b) the method in which those notifications were provided was consistent. The following is a short list of NAS file notification capabilities across the most common platforms (in alphabetical order):

- Dell EMC® Isilon® File Change Notification is limited to SMB shares only. It can be inconsistently applied across shares, meaning that some shares might send notifications while others may not. It can’t send NFS file change notifications.

- Dell EMC VNX® 1/2 and Unity® – Common Event Enabler API (CEE) allows a calling application to receive file event notifications. The API is specific to EMC platforms, and these can change between releases of the operating environment.

- Dell EMC VNX 1/2 File Change Notification (can only be used on pure Common Internet File System (CIFS) environments, and is only supported when the user authentication method is set to NT on the data mover). There are no NFS file change notifications.
• Hitachi® Network-Attached Storage (HNAS) – No file change notification ability or API.

• Hewlett Packard Enterprise® StoreEasy® – Alerts limited to space utilization and quota usage and not file changes.

• NetApp® FPI API, which is most commonly used for distributed hierarchal storage management (DHSM – archiving) purposes.

• Qumulo – No file change notification ability or API.

Clearly, across the major NAS vendors there is no uniformity in terms of their ability to consistently provide notifications related to file change events.

THE PROBLEM WITH CHANGE NOTIFICATION APIS

In the context of synchronizing source and destination systems, file change notification APIs form the basis of an event-based system. As noted earlier, this approach means that whenever a change to a monitored filesystem object is detected, a notification of the event is sent to a monitoring service.

Importantly, when event notifications are sent, there is no acknowledgment process between the sending service and the receiving service, so it will not receive any confirmation that the notification was received. Additionally, the receiving service has no way of determining if a notification was lost in transit. Even a brief network outage or a service restart can cause file change event notifications to be missed.

For the reasons stated above, even if a file change notification API were available for a given platform, it’s fundamentally unreliable.

In conclusion, Change Notification APIs don’t help determine changes in a filesystem prior to copy. The most robust and fault-tolerant method for determining differences between filesystems on two disparate storage platforms is to execute a scan of those filesystems.

LEGACY TOOL SCANS VS. THE DOBIMINER SUITE

Although scanning is clearly the most effective method in determining what file/directory change events have occurred, not all scans are created equal. Some are slow and single threaded, and some can be multithreaded but with no intelligence in how those threads are distributed across the filesystem structure. Filesystem scans with legacy tools result in unacceptably slow performance, given today’s ever-increasing number of files on shared NAS devices.
Unfortunately, the word “scan” has certain historical connotations that should not be applied blindly to new products. A modern scan implementation as found in the DobiMiner Suite is highly efficient compared to the unsophisticated scans executed by legacy tools. In order to understand the different approaches, let’s compare legacy tools like Robocopy and rsync with the DobiMiner Suite.

Rsync has existed since 1996, when a 9 GB hard drive was considered a large capacity storage device. It was originally developed using a single-thread model, since most servers and workstations at the time were single CPU, and it was mainly used for backing up Unix workstations or very small Unix web servers. Fast forward to today and rsync has added many features, but it is still a single-threaded application. It does not take advantage of today’s multicore processors and large memory footprints. Scan performance is therefore extremely slow.

While multiple rsync instances are run in parallel, complicated shell scripts must be written to parse the filesystem structure and assign each portion to a unique instance of rsync. This is a brute force approach that does not scale well and limits performance. Such an approach does not leverage any intelligence regarding the structure, breadth, and/or depth of the filesystem. Valid data could potentially remain unmigrated since scripts will not detect new directories and/or volumes added to the source during the migration.

The tree structure of a filesystem will vary considerably, with some leaves of the tree being narrow and/or shallow, while others are extremely wide and/or deep. In a parallel rsync model, all leaves of the tree have the same weight and will be scanned serially by a single thread. In this model, thread usage is extremely uneven and inefficient.

In Figure 2, the rsync single-threaded model is not highly efficient in the context of a large filesystem with multiple levels of directories. The scan thread has to work its way through the directory tree in a serial fashion. Rsync is also limited to use with filesystems accessed over NFS, so it only covers a portion of modern NAS devices capable of servicing both NFS and SMB clients.

**Rsync Scan Thread Usage**

Fixed scan thread count = 1

---

Figure 2 – rsync scan thread execution
Robocopy was introduced with Windows NT 4.0 in 1997—again, when storage capacity was a fraction of what it is today. Like rsync, Robocopy’s development has continued slowly over the years. While it has evolved to include the ability to run multiple copy threads (8 by default), only a single scan thread is utilized to update filesystem maps.

In Figure 3 below, Robocopy launches a single thread that must serially traverse all directory structures across all filesystems. Robocopy is also limited to scanning NTFS filesystems, so, like rsync, it can only scan a portion of modern NAS devices capable of providing storage access to both SMB and NFS clients.

**Robocopy Scan Thread Usage**

Fixed scan thread count = 1

![Figure 3 – Robocopy scan thread execution](image)

Turning to the DobiMiner suite, scans are significantly faster than legacy tools due to DobiMiner’s ability to not only launch a larger number of threads of execution, but to do so more intelligently.

Conceptually, we create a queue and then execute a number of individual worker threads, with each one pointing to various directories. Figure 4 shows a simplistic illustration of how the DobiMiner Suite scans the filesystem objects contained within each directory and does this in parallel. Each thread is processing portions of the overall filesystem simultaneously. Additionally, since the DobiMiner Suite’s software stack includes both SMB and NFS proxies, both NTFS and Linux filesystems can be scanned in parallel—rsync and Robocopy are limited to either Linux or NTFS filesystems, respectively, and scan the filesystem in a serial fashion.
DobiMigrate/DobiReplicate Scan Thread Usage
Default scan thread count = 64 (threads distributed across running proxies)

Figure 4 – DobiMiner Suite thread execution

SUMMARY

Cross-platform NAS data synchronization requires highly innovative software to produce very fast scan times. The DobiMiner Suite is significantly faster than legacy tools, and works with any NAS platform (including standalone servers). It is extremely accurate, with no dependency on various platform APIs to report and distribute filesystem change notifications.

Additionally, both SMB and NFS are handled with a single tool that requires no scripting. Results provided by highly efficient parallel scans between platforms guarantee delta operations are quickly generated to maintain synchronization between the source and destination.
WOULD YOU LIKE TO KNOW MORE?

Contact sales@datadobi.com

Datadobi.com

Copyright © 2020 Datadobi, All rights reserved.