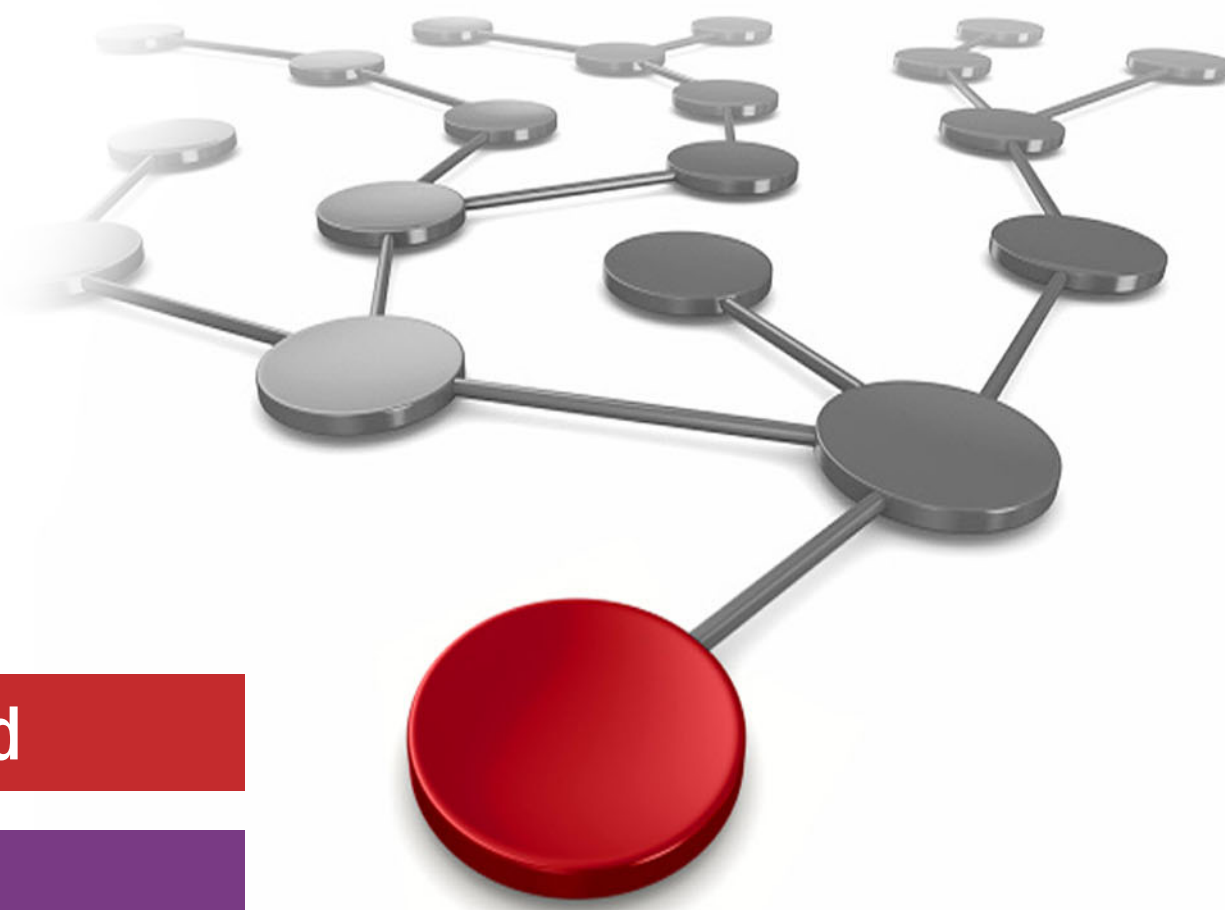


Integration of IBM Aspera Sync with IBM Spectrum Scale Protecting and Sharing Files Globally

Nils Haustein

Jose M Gomez

Benjamin C Forsyth



 Cloud

Storage



IBM Redbooks

Integration of IBM Aspera Sync with IBM Spectrum Scale

March 2019

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (March 2019)

This edition applies to Version 5 of IBM Spectrum Scale (product number 5641-DA1, DA3, or DA5) and Version 3, Release 9 of IBM Aspera (product number 5737-F70).

This document was created or updated on March 29, 2019.

© Copyright International Business Machines Corporation 2019. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too!	viii
Comments welcome	viii
Stay connected to IBM Redbooks	ix
Chapter 1. IBM Aspera Sync and IBM Spectrum Scale overview	1
1.1 IBM Aspera Sync	2
1.2 IBM Spectrum Scale	2
1.2.1 Extended attributes	3
1.3 Differentiation with IBM Spectrum Scale Active File Management	3
1.4 Performance measurements	6
Chapter 2. Setting up IBM Aspera Sync with IBM Spectrum Scale	9
2.1 Overview	10
2.2 Preparation	10
2.3 How async works	11
2.3.1 Async example	11
2.3.2 Sync Database	12
2.3.3 Sessions	13
2.3.4 Replication modes	14
2.3.5 Control overwrites	14
2.3.6 Control deletion on target	14
2.3.7 Using file lists	15
2.3.8 Cluster awareness	16
2.4 How ascp works	17
2.4.1 Ascp example	18
2.4.2 Transfer modes	19
2.4.3 Control overwrites	19
2.4.4 Control deletes on target	20
2.4.5 Different path names on source and target	20
2.4.6 Using file lists and file pair lists	21
2.4.7 Cluster awareness	25
2.5 IBM Spectrum Scale configuration	25
Chapter 3. Using IBM Aspera Sync with IBM Spectrum Scale	27
3.1 IBM Spectrum Scale policy engine considerations with IBM Aspera	28
3.2 Async without policy engine	29
3.3 Ascp without policy engine	30
3.4 Integration with the IBM Spectrum Scale policy engine	30
3.4.1 IBM Spectrum Scale policy engine	31
3.4.2 Integration with async	37
3.4.3 Integration with ascp	41
3.5 Integration with Hierarchical Storage Management	45
3.5.1 Async with HSM on source	46
3.5.2 Async with HSM on target	49

3.6	Integration with immutable files	51
3.7	Integration with IBM Spectrum Scale Snapshots	54
3.7.1	Considerations with ascp and snapshots	56
3.7.2	Using the policy engine with snapshots	57
	Chapter 4. Summary	59
4.1	Async or ascp standalone	60
4.2	Async or ascp integrated with the policy engine	60
4.3	Stand-alone LIST policies or external policy script	60
4.4	Use of snapshots	61
4.5	Use cases	61
4.5.1	File sharing	61
4.5.2	File migration	61
4.5.3	Disaster recovery	62
	Related publications	63
	IBM Redbooks	63
	Online resources	63
	Help from IBM	64

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Redbooks (logo) ®
Aspera®
FASP®
GPFS™

IBM®
IBM Spectrum™
IBM Spectrum Archive™
IBM Spectrum Protect™

IBM Spectrum Scale™
Linear Tape File System™
Redbooks®
Redpaper™

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Economic globalization requires data to be available globally. With most data stored in file systems, solutions to make this data globally available become more important.

Files that are in file systems can be protected or shared by replicating these files to another file system that is in a remote location. The remote location might be just around the corner or in a different country. Therefore, the techniques that are used to protect and share files must account for long distances and slow and unreliable wide area network (WAN) connections.

IBM® Spectrum Scale is a scalable clustered file system that can be used to store all kinds of unstructured data. It provides open data access by way of Network File System (NFS); Server Message Block (SMB); POSIX Object Storage APIs, such as S3 and OpenStack Swift; and the Hadoop Distributed File System (HDFS) for accessing and sharing data.

The IBM Aspera® file transfer solution (IBM Aspera Sync) provides predictable and reliable data transfer across large distance for small and large files. The combination of both can be used for global sharing and protection of data.

This IBM Redpaper™ publication describes how IBM Aspera Sync can be used to protect and share data that is stored in IBM Spectrum™ Scale file systems across large distances of several hundred to thousands of miles. We also explain the integration of IBM Aspera Sync with IBM Spectrum Scale™ and differentiate it from solutions that are built into IBM Spectrum Scale for protection and sharing. We also describe different use cases for IBM Aspera Sync with IBM Spectrum Scale.

Authors

This paper was produced by a team of specialists from around the world working with the IBM Redbooks, Poughkeepsie Center.

Nils Haustein is a Senior Technical Staff Member at IBM Systems group and is responsible for designing and implementing backup, archiving, file, and Object Storage solutions in EMEA. He co-authored the book *Storage Networks Explained*. As a leading IBM Master Inventor, he has created more than 160 patents for IBM and is a respected mentor for the technical community worldwide.

Jose M Gomez is a Senior Software Engineer at IBM Aspera, part of the IBM Hybrid Cloud group. He has 15 years of experience working in different systems management and networking products.

Benjamin C Forsyth is a Director of Core Engineering at IBM Aspera in Emeryville, CA, US. He has 23 years of experience in network applications development. He has worked at IBM for 5 years. His areas of expertise include software development and systems management, specializing in high-speed network data transfer.

Thanks to the following people for their contributions to this project:

Larry Coyne

IBM Redbooks® - IBM Systems Worldwide Client Experience Centers, Poughkeepsie Center

Abhishek Dave

Sandeep R Patil

Venkat Puvvada

Wayne Sawdon

Carl Zetie

IBM Systems

Sudhir Saripalli

Eric Schorger

IBM Hybrid Cloud

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



IBM Aspera Sync and IBM Spectrum Scale overview

This chapter gives an overview of IBM Aspera Sync and IBM Spectrum Scale. It describes how the products complement each other and what differentiates them (IBM Aspera Sync and IBM Spectrum Scale Active File Management Disaster Recovery). Some in-house performance measurements also are available that show the benefits of the use of IBM Aspera Sync for long-distance file transfers.

This chapter includes the following topics:

- ▶ 1.1, “IBM Aspera Sync” on page 2
- ▶ 1.2, “IBM Spectrum Scale” on page 2
- ▶ 1.3, “Differentiation with IBM Spectrum Scale Active File Management” on page 3
- ▶ 1.4, “Performance measurements” on page 6

1.1 IBM Aspera Sync



For all the amazing technological progress that was made in analytics and cloud, the fundamental challenges of reliably transferring and distributing large files and volumes of big data at high speed to locations around the world still persist. In fact, this big data movement problem is more pervasive and daunting across industries with the exponential growth of data that is generated globally.

IBM Aspera solutions are designed to help you globally import, distribute, and synchronize large files and folders directly to and from any major cloud or on-premises storage, without compromising performance or security. Built on IBM Aspera's patented FASP® transfer protocol (which consistently ranks first in every WAN transfer throughput benchmark), IBM Aspera solutions offer secure, scalable capabilities that can grow with your business.

IBM Aspera invested in the integration with IBM Spectrum Scale to perform file transfers between file systems that are in different IBM Spectrum Scale clusters in different locations. The use of IBM Aspera with IBM Spectrum Scale includes the following key benefits:

- ▶ Fast and secure file transfer
- ▶ Predictable and reliable file transfer times
- ▶ Full support for IBM Spectrum Scale extended attributes, including immutability attributes and Access Control Lists (ACL)

1.2 IBM Spectrum Scale



IBM Spectrum Scale is a software-defined scalable parallel file system that provides a comprehensive set of storage services. For more information, see [IBM Knowledge Center](#). It facilitates independent scalability in performance, capacity, and reliability.

The IBM Spectrum Scale file systems is provided by a set of nodes (servers) with internal or external storage that is accessed by way of storage area networks (SAN). Files that are stored in an IBM Spectrum Scale file system can be accessed concurrently from different nodes.

An IBM Spectrum Scale cluster can provide up to 256 file systems, and these file systems can be partitioned into file sets. From a user perspective, a file set is a directory in the file system. From an administrative perspective, a file set can be configured with more functions, such as snapshots, quota, active file management (AFM), and immutability. Therefore, instead of creating a snapshot for an entire file system, the administrator can create a snapshot for a partition of the file system (a file set), which is a directory within the file system.

IBM Spectrum Scale has many features beyond open data access through Network File System (NFS); Server Message Block (SMB); POSIX; Object Storage APIs, such as S3 and OpenStack Swift; and the Hadoop Distributed File System (HDFS). These features include policy-based storage tiering, file encryption and compression, file and command audit logging, and synchronous and asynchronous replication. Based on its modular design, IBM Spectrum Scale file systems scale independently in I/O performance and storage capacity.

An IBM Spectrum Scale cluster consists of a set of nodes that provide access to the underlying storage. This set of nodes can be configured for high availability, which tolerates outages of a minority of nodes.

Although IBM Spectrum Scale provides integrated ways for protecting and sharing files, scenarios exist in which these methods cannot be used. In these cases, IBM Aspera Sync can be integrated with IBM Spectrum Scale to perform the large distance file transfer at the best achievable speed (for more information, see 1.3, “Differentiation with IBM Spectrum Scale Active File Management”).

IBM Spectrum Scale provides extended attributes to store other file system metadata. For example, with NFS version 4 and SMB version 3, IBM Spectrum Scale stores access control lists (ACL) in extended attributes. IBM Aspera Sync version 3.9 and higher fully supports IBM Spectrum Scale extended attributes, which are preserved at the target file system.

1.2.1 Extended attributes

IBM Spectrum Scale at its core provides POSIX-compatible file systems. In addition, it provides extended attributes that include NFSv4 ACLs, immutability flags, retention setting, user-defined attributes, and system-specific attributes. These attributes are not surfaced through the POSIX interface; instead, IBM Spectrum Scale provides an API to access and manages these attributes. For more information, see [IBM Knowledge Center](#).

When a file is copied from a source to a target directory by using standard POSIX commands, these extended attributes of the file are not preserved on the target. Therefore, a special integration is required to preserve the files' extended attributes at the target.

IBM Aspera Sync at version 3.9 and higher incorporates support for IBM Spectrum Scale extended attributes. In particular, IBM Aspera Sync reads the extended attributes of the file in the source directory by using the IBM GPFS™ API call `gpfs_fgetattr()`. It then copies these attributes as a blob to the target system and applies these attributes to the file storage in the target directory by using the GPFS API call `gpfs_fsetattr()`. IBM Aspera Sync also preserves immutability attributes of files that are stored in an immutable file set.

1.3 Differentiation with IBM Spectrum Scale Active File Management

IBM Spectrum Scale provides the Active File Management function (AFM) that can be used to protect and share files globally. One of the AFM versions is AFM for Disaster Recovery (AFM DR), which can be used to replicate files from a source directory to a target directory. For more information, see [IBM Knowledge Center](#).

AFM DR is closely integrated with IBM Spectrum Scale and can be managed and monitored through the IBM Spectrum Scale command-line interface (CLI), graphical user interface (GUI), and the IBM Spectrum Scale management API.

IBM Spectrum Scale AFM DR and IBM Aspera Sync are complementary tools that are used to replicate files between the file systems of two clusters. IBM Spectrum Scale AFM DR is the preferred option for a dual site asynchronous replication solution with IBM Spectrum Scale because it is closely integrated with IBM Spectrum Scale.

Some conditions exist where AFM DR cannot be used, but IBM Aspera Sync can be implemented. IBM Aspera Sync can be preferred over AFM DR in the following use cases:

- ▶ The source directory includes immutable files and their immutability attributes must be preserved in the target directory. Consider the following points:
 - IBM Aspera Sync preserves immutability attributes of files in the target.
 - AFM DR does not yet support immutable file sets.
- ▶ A hierarchical storage management function (HSM) to tier data to other storage media (such as IBM Spectrum Archive™, IBM Spectrum Protect™ for Space Management or IBM Spectrum Scale Transparent Cloud Tiering [TCT]) is configured on the source or the target IBM Spectrum Scale file system or both.

With IBM Aspera Sync IBM Spectrum Scale, policies can be used to exclude migrated files from being replicated to avoid unintended recalls.

AFM DR includes limited support for file systems that are managed by HSM. For more information, see the [IBM Community Wikis](#).

- ▶ The distance between the source and the target system is large and the network connection includes limited bandwidth or noticeable latency.

IBM Aspera Sync is optimized for data transfer across long distances over weak network connections.

AFM DR is *not* optimized for file transfer across long distances over weak networks.

- ▶ The workload that is running in the source directory can cause overloads for AFM DR.

IBM Aspera Sync does not track changes to files and blocks in the source directory in real time and cannot become overloaded by too many changes. However, IBM Aspera Sync transfers entire files. Therefore, even if a small portion of a large file changed, the next transfer copies the entire large file.

AFM DR tracks all of the changes to files and blocks in the source directory in real time and transfers only blocks of a file that changed since the last transfer. This feature helps to save bandwidth, but might cause AFM DR to become overloaded if the changes to files and blocks are frequent.

- ▶ The client is looking for a supported tool to copy files from one IBM Spectrum Scale file system to another. These file systems can be within the same cluster or they can be on different clusters.

IBM Aspera Sync is an officially supported tool for copying files, including their attributes and ACLs from a source directory to a target directory within or across IBM Spectrum Scale clusters.

Although AFM can also be used to transfer files, including their attributes and ACLs from a source directory to a target directory within or across IBM Spectrum Scale clusters, it includes dependencies between the file on the source and on the target. These dependencies can cause limitations for file operations the target. With IBM Aspera Sync, files on source and target are independent.

Consider the following points regarding the characteristics of AFM DR:

- ▶ AFM DR is an integrated component of IBM Spectrum Scale and is included in the Data Management Edition license. When AFM DR is configured, operation and monitoring is included in IBM Spectrum Scale.
- ▶ AFM DR includes integrated processes and tools to facilitate failover and fallback.
- ▶ AFM DR transfers changed blocks and not necessarily the entire file again. For example, if a 1 MB block changes for a 1 GB file that was transferred, only 1 MB is transferred on the next cycle.

- ▶ AFM DR supports the transfer of extended attributes, but does not support the transfer of immutability attributes.
- ▶ AFM DR automatically creates a snapshot that is generated on the source and the target directory, which provides a consistent recovery point.
- ▶ AFM DR supports secure data transfer through Kerberos.
- ▶ AFM DR supports IBM Spectrum Scale compression.
- ▶ AFM DR can be configured for high availability by configuring two or more AFM gateways.
- ▶ AFM DR tolerates network outages and automatically resumes the replication operation if the network is available.
- ▶ AFM DR always runs in the background and tracks changes in the source directory. For I/O intensive workload, this feature might cause contentions in the source file system.
- ▶ AFM DR is not supported on file set where IBM Spectrum Archive, IBM Spectrum Protect for Space Management, or IBM Spectrum Scale Transparent Cloud Tiering perform space management.
- ▶ AFM DR requires file set as source and target directories. A file set is partition of the file system that allows more functions, such as AFM DR. From a user perspective, a file set is a directory; from an administrator perspective, the file set can be configured with more properties.

Consider the following points regarding the characteristics of IBM Aspera Sync:

- ▶ IBM Aspera Sync supports the transfer of extended attributes, including immutability attributes.
- ▶ IBM Aspera guarantees performance, regardless of transfer distance, file size or volume, and network conditions.
- ▶ IBM Aspera uses checksum to assure file integrity.
- ▶ IBM Aspera uses encryption for secure data transfer.
- ▶ IBM Aspera Sync supports directories or file sets where IBM Spectrum Archive or IBM Spectrum Protect for Space Management or IBM Spectrum Scale Transparent Cloud Tiering perform space management. However, this function requires the integration with the IBM Spectrum Scale policy engine to identify files that are not migrated.
- ▶ IBM Aspera Sync does not require file sets on the source and the target directory.
- ▶ IBM Aspera Sync shows the progress of file transfers in the CLI and by way of the optional IBM Aspera Console.
- ▶ IBM Aspera Sync does not create snapshots on the source and the target automatically. It also does not provide consistent recovery points. However, IBM Aspera Sync can use a snapshot as the source for the data transfer, which must be created manually.
- ▶ When used in standard mode, IBM Aspera Sync searches the source directory to identify new and modified files that are subject for transfer from the source to the target directory. IBM Aspera Sync can also be integrated with the IBM Spectrum Scale policy engine, which prevents searching the source directory.
- ▶ IBM Aspera Sync that is integrated with IBM Spectrum Scale can run only on one node for a single transfer session because it is not cluster-aware. It does not provide high availability because the session and transfer fail if the node running.
- ▶ IBM Aspera Sync sessions must be scheduled and monitored by the administrator independent of the IBM Spectrum Scale cluster monitoring.

- ▶ Files that are compressed in the source directory are uncompressed before being transferred and then stored uncompressed in the target directory, unless compression is specified by using the IBM Aspera Sync command.

The following IBM Aspera Sync use cases are key (for more information, see Chapter 4, “Summary” on page 59):

- ▶ For protecting and sharing files over long distances at predictable and reliable transfer times
- ▶ For copying files, including their immutability attributes from one IBM Spectrum Scale file system to another
- ▶ For migrating files from one file system to another by using the checksum and incremental synchronization capabilities of IBM Aspera Sync

1.4 Performance measurements

For demonstration purposes, we conducted performance measurements in which we transferred files from a source IBM Spectrum Scale cluster to a target IBM Spectrum Scale cluster across a long distance. The distance between these clusters was almost 10,000 km (5,700 miles).

During these measurements, we compared the performance and standard deviation of multiple replication sessions between IBM Aspera Sync and the open source tool rsync. For more information about the standard rsync utility, see [this website](#). For more information about the rsync patch that works with GPFS extended attributes, see [this website](#).

The source and target clusters were configured on single nodes that are running on a virtual machine. No performance optimization in IBM Spectrum Scale was done. Each cluster featured one file system. The cluster in San Francisco, California hosted the source file system and the cluster in Frankfurt, Germany hosted the target file system. The network connection between both clusters was based on the IBM internal infrastructure that uses the internet. This network included multiple firewalls.

An overview of the test setup is shown in Figure 1-1.

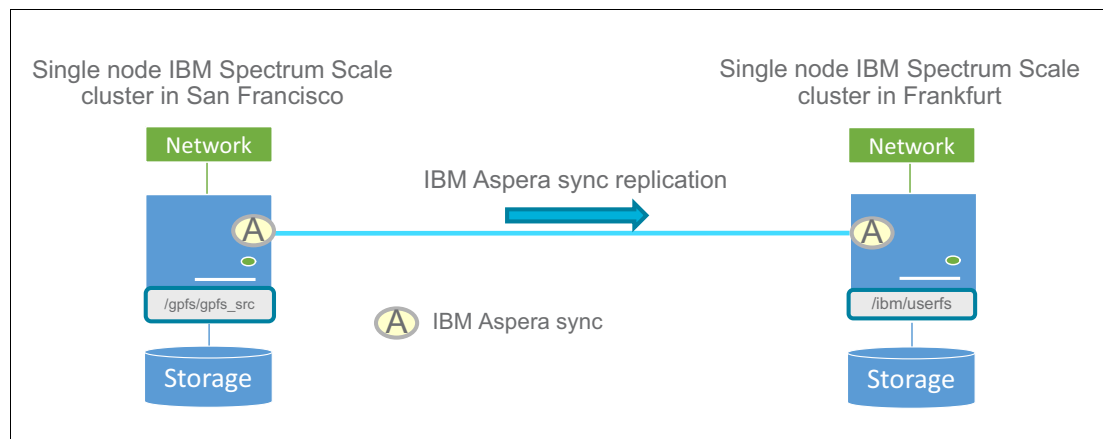


Figure 1-1 Test setup

Different test cases were run several times by using IBM Aspera’s async tool and the open source rsync tool. Each test case transferred several files with a constant file size per test case.

As listed in Table 1-1, the first test case transferred several of files with a size of 256 MB. By using async, this transfer took 48 seconds on average; by using rsync, the transfer took 248 seconds. Therefore, async was five times faster. In addition, the standard deviation of different test runs was only 2 seconds with async and 142 seconds with rsync.

Table 1-1 Test results of Internal measurements of file transfers between California and Germany

Test case	Async duration	Rsync duration	Speedup
256 MB files	48s (std dev 2 s)	4m 8s (std dev 142 s)	5x
512 MB files	1m 16s (std dev 4 s)	9m 41s (std dev 224 s)	7x
1024 MB files	2m 7s (std dev 6 s)	27m 10s (std dev 192 s)	12x
2048 MB files	3m 55s (std dev 11 s)	56m 57s (std dev 497 s)	14x

A plot of the performance measurements that correlate the file size and the transfer duration is shown in Figure 1-2.

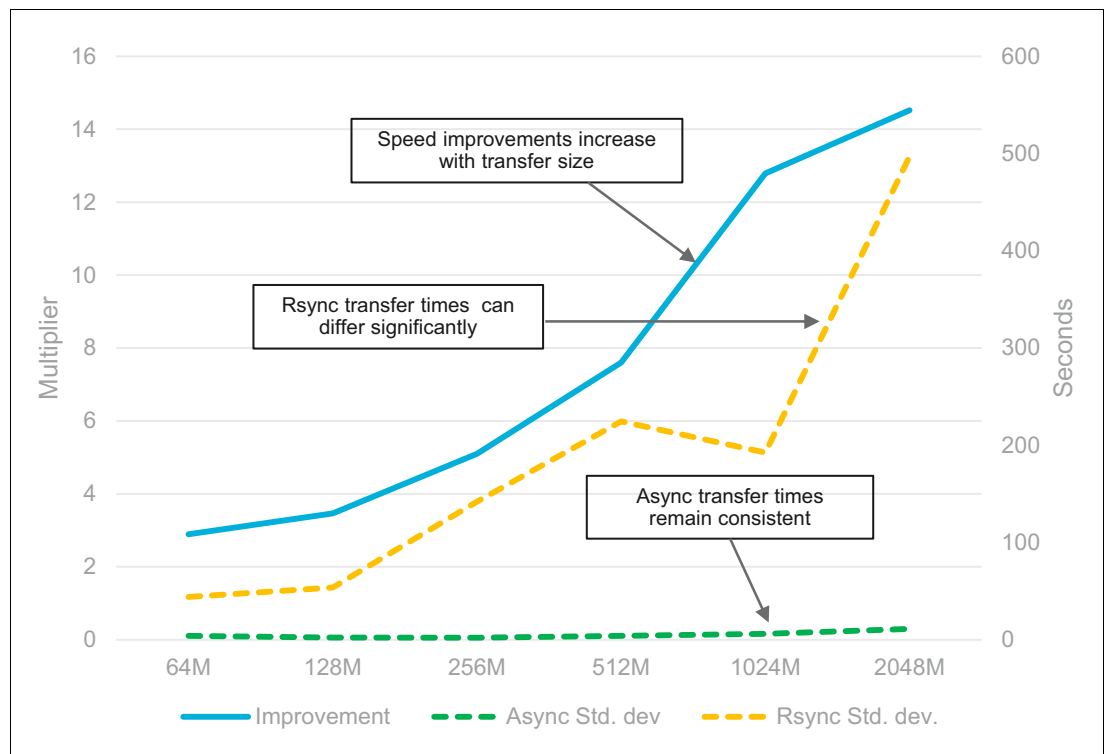


Figure 1-2 Async versus rsync standard deviation of transfer size and time

In summary, the transfer speed of async improves almost linearly with increasing file sizes while rsync can differ significantly. The transfer times that were measured with async are consistent with a low standard deviation, unlike rsync.

In addition, IBM Aspera Sync supports the transfer of immutable files and their immutability attributes; rsync does not.



Setting up IBM Aspera Sync with IBM Spectrum Scale

In this chapter, we describe how the IBM Aspera tools can be integrated with IBM Spectrum Scale. Also described is the integration with the IBM Spectrum Scale policy engine.

This chapter includes the following topics:

- ▶ 2.1, “Overview” on page 10
- ▶ 2.2, “Preparation” on page 10
- ▶ 2.3, “How async works” on page 11
- ▶ 2.4, “How ascp works” on page 17
- ▶ 2.5, “IBM Spectrum Scale configuration” on page 25

2.1 Overview

IBM Aspera Sync version 3.9 and higher supports IBM Spectrum Scale extended attributes and ACL. With support, IBM Aspera Sync transfers NFSv4 ACLs, user-defined attributes, and immutability attributes for files and directories from the source file system and applies these attributes to the files and directories in the target file system. The source and target systems must be IBM Spectrum Scale clusters that are running the same level of IBM Spectrum Scale and IBM Aspera Sync.

IBM Aspera Sync includes two tools that are described in this paper: `async` and `ascp`. `Async` is the comprehensive file synchronization solution (see 2.3, “How `async` works” on page 11). It tracks the files and directories it replicated in a pair of Sync databases that is stored in the source and the target file system. This information is used during the `async` session to determine which files must be transferred. It also contains checksum information for files that were transferred. The underlying transfer of files is done by the `ascp` tool (see 2.4, “How `ascp` works” on page 17).

`Ascp` transfers files from the source to the target directory by using IBM Aspera’s innovative file transfer techniques. `Async` uses `ascp`’s management API to send and receive files. `Ascp` does not identify any files to be transferred; instead, it receives the file names that are to be transferred as input parameter through its management API interface.

Note: The `async` tool is the central part of IBM Aspera Sync and provides the capability to asynchronously replicate the data between a source and target directory. It includes all of the intelligence to identify the files that were created, changed, renamed, or deleted and orchestrate the transfer by using `ascp` in the backend.

2.2 Preparation

To use IBM Aspera Sync with IBM Spectrum Scale, the IBM Aspera Sync software packages must be installed and configured on one or more IBM Spectrum Scale cluster nodes of the source and the target cluster. In addition, the user’s SSH-keys that are used for the file transfer must be exchanged between the source and the target nodes that are involved in the file transfer.

IBM Aspera Sync can be installed on one or more nodes of the source and the target cluster. One IBM Aspera Sync session that replicates files from a source directory to a target directory is normally established between a pair of IBM Spectrum Scale nodes, including one node from the source cluster and one node from the target cluster. If multiple IBM Aspera Sync sessions are run at the same time, multiple pairs of nodes can participate in this transfer whereby each node pair runs the transfer for one pair of source and target directories.

Note: A single IBM Aspera Sync session can run only on one cluster node of the source cluster to one cluster node of the target cluster.

IBM Aspera Sync can also be used to transfer files within one cluster. In this case, the source and target directories can be within the same IBM Spectrum Scale file system or in different file systems. However, source and target directories must be distinct and cannot be embedded in each other.

2.3 How async works

IBM Aspera Sync is started by using the **async** command that is shown in Example 2-1. Each run of the **async** command is an **async** session. Each **async** session is denoted by a session name. For each session name, **async** creates and maintains a Sync database in the source and target system that contains information about the file attributes, time stamps, and checksums. This database is used by **async** to determine whether a file must be transferred completely, partially, or not at all (see 2.3.2, “Sync Database” on page 12).

After it is started, **async** examines the specified source directory (local directory) and identifies all files and directories that were created, deleted, renamed, and modified since the last **async** run for this session by evaluating the time stamps of the files and matching this information to the Sync Database.

Files are transferred entirely; **async** does not perform changed blocks tracking to transfer only changed blocks. For the file transfer, **async** uses **ascp** in the backend. For files or directories where only the metadata changed, such as ACL, permissions or extended attributes, **Async** does not transfer the entire file but only the changed metadata.

Note: **Async** transfers entire files only. If small changes are made on large files on the source, the entire large file is transferred again. This process can cause extended transfer times and increased load on the network. In such scenario where large files are changed more frequently on the source, it is recommended to consider the use of IBM Spectrum Scale AFM because it transfers changed blocks only.

If during a transfer session the network that is connecting the source and target system fails, **async** can also fail. However, the next session picks up where the previous session failed because the status of the file transfers is tracked in the Sync Database.

2.3.1 Async example

In this section, we consider an **async** command line example where files from the directory `/fs1` of the source cluster are replicated to the directory `/target/fs1` of the target cluster. The host name of the remote cluster node in the target cluster is `remotehost` in this example and the data is transferred by using the privileges and SSH-key of the user-named user. The IBM Spectrum Scale extended attributes, time stamps user, and group ID are preserved in the target directory during this transfer session (see Example 2-1).

Example 2-1 async command example where files from source cluster are replicated to target cluster

```
# async -N sync_fs1 -d /fs1 -r user@remotehost:/target/fs1
--preserve-xattrs=native -i ~/.ssh/id_rsa --preserve-uid --preserve-gid
--preserve-access-time [--preserve-time --create-dir]
```

Consider the following important options:

<code>-N session-name</code>	Session name (must be the same for a specific replication pair)
<code>--preserve-xattrs=native</code>	Transfers extended attributes (required for IBM Spectrum Scale)
<code>--preserve-uid</code>	Preserve user ID
<code>--preserve-gid</code>	Preserve group ID
<code>--preserve-access-time</code>	Preserve last access time stamp (important for immutable files)
<code>--preserve-time</code>	Preserve time stamps for mtime

`--create-dir` Create the destination directory if it does not exist

The `async` command includes the session name `sync_fs1` and transfers files from the source directory `/fs1` to the target directory `/target/fs1`. The source directory is specified with parameter `-d` and can be an IBM Spectrum Scale file system, a file set, or a normal directory and the target directory with parameter `-r`. When file sets are used as the source and must be preserved as file sets on the target, the file sets on the target must be manually created before the replication session is started.

An IBM Spectrum Scale file set is partition of the file system that allows more functions, such as immutability. From a user perspective, a file set is a directory. From an administrator perspective, the file set can be configured with more properties.

Unlike a normal directory, a file set is created and configured by using a set of IBM Spectrum Scale commands. For example, use the following command to create a file set that is named `fsetname` in file system that is named `fname`:

```
# mmcrfileset fname fsetname
```

Use the following command to link this file set to a directory in the file system:

```
# mmlinkfileset fname fsetname -J directory
```

Use the following command to make the file set immutable:

```
# mmchfileset fname fsetname --iam-mode compliant
```

Important: Async does not preserve file sets on the target. These file sets must be manually created on the target.

The user ID, group ID, time stamps, and IBM Spectrum Scale extended attributes are preserved on the target.

The target directory is specified by using parameter `-r` and includes the user and host name of the target cluster. The user and host names can also be given by using the following parameters:

```
--user username
```

```
--host hostname
```

In the next sections, we discuss other parameters that can be used to control the replication sessions and provide best practices guidance in the context of IBM Spectrum Scale.

2.3.2 Sync Database

Each `async` session includes a name that is denoted by the `async` command line parameter `-N` (for more information, see 2.3.1, “Async example” on page 11). For each session name, `async` creates a Sync Database (`snap.db`) that is stored on the source and the target file system.

The database records the state of the file system at the end of the last `async` session. The next time the session is run with the same name, the file system is compared to the database to identify changes.

The Sync database is created in private directories at the root level of the synchronized directory. Async stores file state and checksum information and in-progress transfers (a transfer cache for pending files) in the Sync Database.

The default location of the Sync Database is within the source and target directories. In our example, the database is stored in `/fs1/.private-asp` on the source directory and in `/target/fs1/.private-asp` on the target. Therefore, it is stored within the shared IBM Spectrum Scale file system, which is accessible by all cluster nodes.

The location of the Sync Database in the source and the target can also be changed by using the following parameters:

```
--local-db-dir=lbdbr
```

```
--remote-db-dir=rdbdir
```

Note: If the session name for a specific pair of source and target directories to be synchronized remains the same, the same instance of the Sync database is used.

Async creates and maintains a Sync database for each session name. The location of the Sync Database that is controlled by the parameters `--local-db-dir` and `--remote-db-dir=rdbdir` for a specific session name and pair of source and target directory name must be the same. Also, the Sync Database must be stored in a shared IBM Spectrum Scale file system that is accessible by all nodes that run the `async` command.

One `async` session is run on only one IBM Spectrum Scale cluster node. The next transfer session can be run on a different node, which requires the Sync Database to be available on a shared file system.

2.3.3 Sessions

Each `async` session must include a name that is specified with the `-N` parameter of the `async` command (see 2.3.1, “Async example” on page 11). Async creates and maintains a Sync Database for each session name in the source and target file system.

The session name describes a replication relation between a source and target directory that might be maintained over long periods. The content of this database is queried and updated during an `async` session. Therefore, subsequent `async` sessions with the same session name use the same pair of databases (source and target) and existing file states and checksums.

Note: Do not change the name of the session for a replication relation because changing the session name creates a Sync database and it does not use the information of the old Sync database.

The session name can be understood as an identifier for a replication relation. A replication relation exists between a source and a target directory.

The session name for a replication relation always is the same. If the session name for a replication relation is changed, `async` recalculates the checksum information for all files in the source and target directory again to create a pair of databases.

If files on the source or target are migrated by IBM Spectrum Archive EE, IBM Spectrum Protect for Space Management, or IBM Spectrum Scale Transparent Cloud Tiering, all files are recalled.

2.3.4 Replication modes

Async has three modes of synchronization: **push**, **pull**, and **bid**. The mode can be specified by using the following parameter:

```
-K push|pull|bid
```

With **push** mode, the contents of source directory are synchronized to the target directory whereby contents of the target directory are overwritten with the contents of the source directory (which is default behavior; overwrites can also be controlled). Therefore, the source directory content is pushed to the target directory while target directory contents that did not exist on the source are preserved on the target. This mode is the default mode.

With **pull** mode, the contents of the target directory are synchronized to the source directory whereby contents of the source directory are overwritten with the contents of the target directory (which is the default behavior; overwrites can also be controlled). Therefore, the target directory content is pulled to the source directory while preserving any source files that might not exist in the target directory.

With **bid** (bidirectional) mode the contents of source and target directories are synchronized, with newer versions of files and directories overwriting older versions in the source or target directory (which is the default behavior; overwrites can also be controlled). Because the source and target must be writable, the replication cannot be done from snapshots. When immutable file sets are used on the source and target, conflicts can occur when file names on the source and target are the same.

In most cases, the **push** mode (default) is used when async is used to transfer files from an IBM Spectrum Scale file system.

2.3.5 Control overwrites

The behavior of overwriting files in the target directory can be controlled by using overwrite policies, as shown in the following example:

```
--overwrite=always|older|conflict
```

The value of **always** means that a file on the target is overwritten if it was modified on the source. This value is default value.

The value of **older** means that the file on the target is overwritten if it is older than on source. When the **--overwrite=older** option is used, the command option **-t (--preserve-time)** must be used with the **async** command.

The value of **conflict** is applicable for bidirectional replication mode only.

In most cases, the default value of **always** is sufficient.

2.3.6 Control deletion on target

By default, async deletes files in the target directory that were deleted in the source directory during the async session. Alternatively, async can also be configured to not delete any files in the target directory, even though files were deleted in the source directory by using the following parameter:

```
--ignore-delete
```

Alternatively, the `--delete-delay` parameter can be used to postpone deleting files or directories until the end of the async session.

In most cases, deleting files in the target directory during the replication session is sufficient. Some rare cases exist where files on the target must not be deleted (use parameter `--ignore-delete`). However, this use causes that the target directory to use more storage capacity than the corresponding source directory.

2.3.7 Using file lists

Async allows transferring files with their names that are specified in file list. This feature is useful when IBM Aspera Sync is integrated with the IBM Spectrum Scale policy engine (see Chapter 3, “Using IBM Aspera Sync with IBM Spectrum Scale” on page 27). Include and exclude statements can be used to specify the file names to be included or excluded in the file transfer performed by async.

For more information about include and exclude filtering rules, see [this website](#).

When include and exclude lists are provided, async examines the source directory and identifies files that match include and exclude patterns. Then, it checks whether the files were modified since the last async run.

Use the following parameter to include files with their names that are provided in a file list:

```
--include-from=filelist
```

Use the following parameter to include individual file names:

```
--include filename
```

Individual file names or file names that are provided in a file list also can be excluded by using the following parameters:

```
--exclude-from=filelist
```

```
--exclude filename
```

File lists, including file and directory names to be included, must be strictly sorted and include separate directory and file names that are relative to the source directory. Creating a file list for the files and directories that are stored in the source directory named `/fs1` is shown in Example 2-2.

Example 2-2 File list including file and directory names to be included

```
/fs1/file1  
/fs1/dir1/file1  
/fs1/dir1/file2  
/fs1/dir2/file3  
/fs1/dir2/dir3/file4
```

The resulting file list that is expected by async shown in Example 2-3. The directory and file names are separated and relative to the source directory `/fs1`.

Example 2-3 Resulting file list expected by async

```
file1  
dir1/  
dir1/file1
```

```
dir1/file2
dir2/
dir2/file3
dir2/dir3
dir2/dir3/file4
```

Include and exclude rules are applied in the order that they are encountered. The first matching rule (whether including or excluding) takes precedence.

Many more include and exclude samples that use pattern matching can be found in the *async man* page and the [IBM Aspera High-Speed Transfer Server Admin Guide 3.9.1](#).

Example 2-4 shows an *async* command that includes files with their names that are stored in a file list (*filelist*) and excludes everything else. The file list must have the include file and directory names, as shown in Example 2-3 on page 15.

Example 2-4 Async command that includes files with their names stored in a file list (filelist)

```
# async -N sync_fs1 -d /fs1-r user@remotehost:/target/fs1 -l 1G -K push -i
~/ssh/id_rsa --preserve-xattrs=native -u -j -t --preserve-access-time
--include-from=filelist --exclude "*" 
```

Note: The order of the include and exclude parameters in the command that is shown in Example 2-4 is important for the matching of the file names.

If the order of the parameters `--include-from=filelist --exclude "*"` are switched, every file is excluded.

Transferring files with their names that are given in a file list is useful when integrating the *async* command with the IBM Spectrum Scale policy engine (see 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30).

2.3.8 Cluster awareness

In this section, we describe cluster awareness. A *cluster* is a set of nodes. In typical IBM Spectrum Scale configurations, multiple nodes exist that provide access to the global file systems. Programs, such as *async*, can be installed and run simultaneously on one or more nodes of the cluster.

Cluster awareness means whether it is feasible to run multiple instances of *async* under the same session name simultaneously on multiple IBM Spectrum Scale nodes. The answer to this question is no.

Async uses the local and remote database to track the status of files and select files for transfer that is based on the checksums. Therefore, each *async* process must have access to this database.

If *async* runs simultaneously on multiple nodes and replicates data for the same session, these *async* processes access the same database from different nodes. Concurrent access to the Sync DB from different nodes is untested and not recommended. It *is* recommended to run *async* for a specific *async* session on one node.

To accelerate the transfer time, multiple transfer threads can be used on a single node. Running a later replication session with the same name on another nodes is also possible, if the database is stored in a shared file system. The concurrent access to the Sync DB from different nodes can result in an error being generated by async because it sees that the process on the other node includes a file lock that is taken out on the database.

Important: It is not recommended to run parallel async processes on more than one node for the same async session name.

Async cannot run on ESS I/O nodes. Async can be run on dedicated nodes or on protocol node, which provides NFS, SMB, Object, or HDFS access.

2.4 How ascp works

Ascp is the backend of async and performs the file transfer from the source to the target by using IBM Aspera's leading file transfer techniques. For more information, see [this website](#). Ascp does not identify files that are subject for transfer; instead, it takes the file names as the input parameter and transfers these files to the specified target directory. Ascp transfers files entirely.

Ascp can be started independent of async. It does not require the Sync Database. When async is used after ascp is used for the same pair of source and target directories, async does not know about the metadata of the files that were transferred by ascp. Therefore, async must inventory all of the files that were copied by ascp and determine their checksum and state in the source and target directory.

Note: The combination of the use of ascp with async and async must be carefully planned because ascp transfers files without providing information about these files to async. Use cases might exist where ascp is run periodically, such as multiple times a day to copy files from the source to the target. Async can be run less frequently (such as once per week on the weekend) to assure consistency in case ascp was not successful copying files during the daily runs.

When ascp is used without async, the default mode of ascp is to transfer files and file extended attributes regardless of what changed on the file. If only the extended attributes of a file were changed by the file owner, ascp applies these attributes only if the file is forced to transfer by specifying the `-overwrite=always` option.

Note: If only extended attributes changed on a file in the source directory and the file is transferred by using ascp, it transfers the entire file and applies changes of extended attributes to the file on the target directory.

If during an ascp transfer job the network that is connecting the source and target system fails, ascp also might fail. Because ascp does not track the status of the file transfer in the Sync database, the next ascp transfer job copies all of the files again that are specified in the file list.

2.4.1 Ascp example

In the ascp example that is shown in Example 2-5, the contents of the source directory /fs1 is copied to the target directory /target/fs1 of the remote cluster. For more information, see [this website](#).

Example 2-5 Copying contents of source directory to target directory of remote cluster

```
# ascp -l 1G -d -p --preserve-xattrs=native --preserve-file-owner-gid
--preserve-file-owner-uid -i ~/.ssh/id_rsa /fs1/ user@remotehost:/target/fs1/
```

The host name of the remote cluster node is remotehost in Example 2-5 and the data is transferred by using the privileges and SSH-key of the user named user. The IBM Spectrum Scale extended attributes, time stamps user, and group ID are preserved.

The following options are available:

- ▶ --preserve-file-owner-uid: Preserve user ID
- ▶ --preserve-file-owner-gid: Preserve group ID
- ▶ --preserve-xattrs=native: Preserve extended attributes (required for IBM Spectrum Scale)
- ▶ -p: Preserve time stamps (important for immutable files)
- ▶ -d: Create destination directory if this does not exist

The source directory (/fs1) can be an IBM Spectrum Scale file system, a file set, or a normal directory, and the target directory. When file sets are used as the source and must be retained on the target, the file sets on the target must be manually created before the replication session is started.

The user ID, group ID, time stamps, and IBM Spectrum Scale extended attributes are preserved on the target. Ascp also allows copying single files from the source directory.

An IBM Spectrum Scale file set is partition of the file system that allows more functions, such as immutability. From a user perspective, a file set is a directory. From an administrator perspective, the file set can be configured with more properties.

Unlike a normal directory, a file set is created and configured by using a set of IBM Spectrum Scale commands. For example, use the following command to create a file set that is named fsname in file system that is named fsname:

```
# mmcrfileset fsname fsetname
```

Use the following command to link this file set to a directory in the file system:

```
# mmlinkfileset fsname fsetname -J directory
```

Use the following command to make the file set immutable:

```
# mmchfileset fsname fsetname --iam-mode compliant
```

Note: Ascp does not preserve file sets on the target. These file sets must be manually created on the target.

In our example, the target directory notation includes the user and host names of the target cluster. The user and host names can also be given by using the following parameters:

```
--user username
--host hostname
```

Next, we discuss other parameters that can be used to control the ascp.

2.4.2 Transfer modes

Ascp can be configured in the following transfer modes:

```
--mode=send|recv
```

With **send**, ascp sends the files from the source to the target directory. This action is the default. With **recv**, ascp receives files from the target into the source directory. When **--mode** is used, the **--host** parameter must be specified. In most cases, the mode **send** is used.

2.4.3 Control overwrites

Overwriting files that are on the source and target can be controlled by using the following parameter:

```
--overwrite={never|always|diff|diff+older|older}
```

The value of **never** specifies that files on the target are not overwritten. However, if the parent folder on the target is not empty, its time stamps (access, modify, and change times) can still be updated.

The value of **always** specifies that files on the target are always overwritten.

The value of **diff** means that the file on the target is overwritten if it is different from the source. If a complete file at the target is the same as a file on the source, it is not overwritten. Partial files are overwritten or resumed depending on the resume policy. This value is the default.

The value of **diff+older** means that the file on the target is overwritten if it is older *and* different than the source file. For example, if the file on the target is the same as the source, but with a different time stamp, it is not overwritten. Also, the file on the target is not overwritten if it is different than the source but newer.

The value of **older** specifies that the file on the target is overwritten if its time stamp is older than the source time stamp (**mtime**).

The overwrite methods **diff** or **diff+older** can be further controlled with the resume policy (parameter **-k**). Consider the following points:

- ▶ If **-k 0** or no **-k** is specified, the source and target files are always considered different and the target file is always overwritten.
- ▶ If **-k 1**, the source and target files are compared based on file attributes (currently file size).
- ▶ If **-k 2**, the source and target files are compared based on sparse checksums.
- ▶ If **-k 3**, the source and target files are compared based on full checksums.

In general, the use of the overwrite method **diff** or **diff+older** assures that only files are transferred that changed in the source directory. This method can help to save network resources. When the resume policy is used, calculating the checksums takes more time.

2.4.4 Control deletes on target

By default, `ascp` does not delete files in the target directory that were deleted in the source directory. To delete files in the target that were deleted on source, use the following parameter:

```
--delete-before-transfer
```

By using this parameter, `ascp` identifies and deletes files on the target that do not exist on the source before transferring files from the source to the target directory. This condition applies only to file names that are provided to `ascp`. The `asdelete` tool provides the same capability.

The `asdelete` tool compares the source directory with the target directory and deletes extraneous files from the target directory. The high-level syntax is shown in Example 2-6.

Example 2-6 Asdelete tool high-level syntax

```
# asdelete --host remotehost--auth-name username --auth-pass password  
/source_directory /target_directory
```

Notice that `asdelete` follows symbolic links, which can result in files being deleted that are not within the target directory.

2.4.5 Different path names on source and target

When `ascp` is used, the content of the source path is copied to the specified path on the target, including the trailing path name of the source directory. For example, the source path `/fs1/Documents` can be copied to the target path `/target/fs1` by using the command that is shown in Example 2-7.

Example 2-7 Copy source content to target path

```
# ascp -l 1G -d -i id-file --overwrite=diff -k 2 --mode=send -p  
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid  
/fs1/Documents/ user@remotehost:/target/fs1
```

The resulting directory structure on the target includes the trailing path name of the source, as listed in Table 2-1.

Table 2-1 Resulting target path names

On Source	On Target without <code>--src-base=/fs1/Documents</code>
<code>/fs1/Documents/file1</code>	<code>/target/fs1/Documents/file1</code>
<code>/fs1/Documents/dir1/file2</code>	<code>/target/fs1/Documents/dir1/file2</code>
<code>/fs1/Documents/dir2/file3</code>	<code>/target/fs1/Documents/dir2/file3</code>

It is also possible to strip specified path prefixes from the source path of each transferred file or directory.

Consider the following scenario: The source path to be copied is `/fs1/Documents/`. The content (subdirectories and files) of the source path must be copied to the target path `/target/fs1`. The path name of the files that are copied to the target must be relative to the `/fs1/Documents` source path.

Therefore, the directory Documents must not appear in the target folder. To achieve this configuration, the following ascp command line option can be used:

```
--src-base=source-path-prefix
```

The use of this command strips the specified source-path prefix from the source path of each transferred file or folder. The remaining portion of the path remains intact at the target.

For example, to transfer the directories and files that are stored in the source directory /fs1/Documents by eliminating the directory name /Documents in the target directory, set the --src-base=/fs1/Documents by using the ascp command that is shown in Example 2-8.

Example 2-8 Command without /Documents in the target directory

```
# ascp -l 1G -d -i id-file --overwrite=diff -k 2 --mode=send -p
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid
--src-base=/fs1/Documents /fs1/Documents user@remotehost:/target/fs1
```

As a result, the source directories and files appear as listed in Table 2-2.

Table 2-2 Results of source directories and files

On Source	On Target with --src-base=/fs1/Documents	On Target without --src-base=/fs1/Documents
/fs1/Documents/file1	/target/fs1/file1	/target/fs1/Documents/file1
/fs1/Documents/dir1/file2	/target/fs1/dir1/file2	/target/fs1/Documents/dir1/file2
/fs1/Documents/dir2/file3	/target/fs1/dir2/file3	/target/fs1/Documents/dir2/file3

Note: Sources that are outside of the source base are not transferred. Although no errors or warnings are issued, the skipped files are logged. For example, if /fs1/file4 was included in our example sources, it is not transferred because it is outside of the specified source base (/fs1/Documents/).

Stripping off path name prefixes can also be used with file list or file pair lists. For more information, see “File lists” on page 22.

Stripping the path prefixes of files that are provided in a file list is useful when integrating the ascp command with the IBM Spectrum Scale policy engine. For more information, see 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30.

2.4.6 Using file lists and file pair lists

Ascp can also transfer files with their names that are provided in file lists. Two options are available to use file lists with ascp: file list, including the source path and file names (for more information, see , “File lists” on page 22), and file lists that include pairs of source and target path and file names (for more information, see “File pair lists” on page 23).

It also is possible to strip off a specified prefix of the source path when files are stored in the target directory. For more information, see 2.4.5, “Different path names on source and target” on page 20).

File lists

To use ordinary file lists, including the fully qualified path and file names of the files to be transferred from the source directory, the following parameter can be used:

```
--file-list=filelist
```

The `filelist` file is a text file that includes the fully qualified path and file names of the files to be transferred from the source to the target. Each source file is specified on a separate line. UTF-8 file format is supported. Only the files and directories are transferred (path information is not preserved at the destination).

To read a file list from standard input, use “-” in place of file, as shown in Example 2-9 in which `filelist` contains the list of source files and directories.

Example 2-9 List of source files and directories

```
/tmp/code/compute.php  
doc_dir/  
images/iris.png  
images/rose.png
```

Then, the command that is shown in Example 2-10 is run.

Example 2-10 Run the ascp command

```
# ascp -l 1G -d -i id-file --overwrite=diff -k 2 --mode=send -p  
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid  
--file-list=list.txt --user=user --host=remotehost /
```

The target, which is the transfer user’s docroot (that is, a configuration option that specifies the area of the file system to which the user can access), contains the output that is shown in Example 2-11.

Example 2-11 Command results

```
compute.php  
doc_dir/ (and its contents)  
iris.png  
rose.png
```

All directory prefixes of the path and file names on the source are removed on the target. To keep path names on the target, use the `--src-base` parameter. For example, to keep the directory structure of the files in the file list on the target, use the command that is shown in Example 2-12.

Example 2-12 Run ascp command

```
# ascp -l 1G -d -i id-file --overwrite=diff -k 2 --mode=send -p  
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid  
--src-base=/ --file-list=list.txt --user=user --host=remotehost /
```

On the target, the directory structure is identical to the source, as shown in Example 2-13.

Example 2-13 Target directory structure is identical to the source

```
/tmp/code/compute.php  
doc_dir/
```

images/iris.png
images/rose.png

Note: The files and directories in the filelist must be within the `--src-base` value of `/fs1/Documents` on the source. Files that do not match the `--src-base` path prefix are not copied.

If the source and target directory and file names are identical, the `--src-base` parameter and the target directory must be set to `/` while the file list includes the fully qualified path names that are shown in Example 2-14.

Example 2-14 Parameters to use when source and target directory and file names are identical

```
# ascp -l 1G --file-list=filelist --overwrite=diff -k 2 --mode=send  
--user=user --host=remotehost -i id_file -p --preserve-xattrs=native  
--preserve-file-owner-gid --preserve-file-owner-uid  
--src-base=/ /
```

Consider the following restrictions when file lists are used with `ascp`:

- ▶ The command line cannot use the `user@host:source` syntax. Instead, specify this information by using the `--mode`, `--host`, and `--user` options.
- ▶ Paths that are specified in the file list cannot use the `user@host:source` syntax.
- ▶ Because multiple sources are transferred, the destination must be a directory.
- ▶ Only one `--file-list` or `--file-pair-list` option is allowed per `ascp` session. If multiple lists are specified, only the last one is used.
- ▶ Only files and directories that are specified in the file list are transferred; any sources that are specified on the command line are ignored.
- ▶ If the source paths are URLs, the size of the file list cannot exceed 24 KB.

File pair lists

`Ascp` can also transfer files with the file and path names in the source and target directory that are provided as file pair in a file pair list. To provide a file pair list to `ascp`, use the following parameter:

```
--file-pair-list=filepairlist
```

The file `filepairlist` is a text file that includes the file and directory names of the files to be transferred from the source followed by the file and directory name on the target. Each source and target file name is specified on a separate line. The UTF-8 file format is supported.

Note: The target file name is relative to the transfer user's docroot. Docroot (or absolute path) is a configuration option and specifies the area of the file system that is accessible to an IBM Aspera transfer user. The default empty value allows access to the entire file system. Even if a target name is specified as an absolute path, the resulting path at the target is still relative to the docroot. Target paths that are specified in the list are created automatically if they do not exist.

For example, the `filepairlist` contains the list of sources (odd line numbers) and destinations (even line numbers) that are shown in Example 2-15.

Note: The line numbers that are shown in Example 2-15 are given to denote odd and even lines and must be omitted for the transfer.

Example 2-15 Directory and file names for filepairlist

```
1. Dir1
2. Dir2
3. my_images/iris.png
4. project_images/iris.png
5. /tmp/code/compute.php
6. /tmp/code/compute.php
7. /tmp/tests/testfile
8. testfile2
```

Then, the command that is shown in Example 2-16 is run.

Example 2-16 Run the ascp command

```
# ascp -l 1G -d -i id-file --overwrite=diff -k 2 --mode=send -p
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid
--file-pair-list=filepairlist --mode=send --user=user --host=remotehost /
```

The destination (in this case, the transfer user's docroot) now contains the output that is shown in Example 2-17.

Example 2-17 Destination of the transfer of user's docroot

```
Dir2 (and its contents)
project_images/iris.png
tmp/code/compute.php
testfile2
```

Directory Dir2 on the target contains the content of the directory Dir1 of the source. Likewise, the source file my_images/iris.png is named project_images/iris.png on the target.

Consider the following restrictions when file pair lists are used:

- ▶ The command line cannot use the user@host:source syntax. Instead, specify this information by using the --mode, --host, and --user options.
- ▶ The **user@host:source** syntax cannot be used with paths specified in the file list.
- ▶ Because multiple sources are being transferred, the destination that is specified on the command line must be a directory.
- ▶ If the file pair list contains the absolute path of the target files and directories, the destination directory should be set to /.
- ▶ Only one --file-pair-list or --file-list option is allowed per ascp session. If multiple lists are specified, only the last one is used.
- ▶ Only files from the file pair list are transferred; any other source files that are specified on the command line are ignored.
- ▶ If the source paths are URLs, the file list cannot exceed 24 KB.

The use of file lists that describe the file names to be transferred is useful when integrating the ascp command with the IBM Spectrum Scale policy engine. For more information, see 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30.

2.4.7 Cluster awareness

An IBM Spectrum Scale cluster consists of multiple cluster nodes. Cluster awareness means whether it is feasible to run multiple instances of `ascp` simultaneously on multiple IBM Spectrum Scale nodes.

Running `ascp` for the same source directory in parallel on multiple nodes does not include the limitation for `async` that is described in 2.3.8, “Cluster awareness” on page 16). This limitation does not exist because `ascp` does not use the Sync Database; therefore, it does not need to access a database simultaneously from multiple nodes.

When running `ascp` simultaneously on multiple nodes, it must be assured that the `ascp` instances do not transfer the same files. Transferring the same files can result in file access conflicts. Running `ascp` on multiple nodes for the same source directory and different subsets of files can be automated through the integration of the IBM Spectrum Scale policy engine. For more information, see 3.4.3, “Integration with `ascp`” on page 41.

Similar to `async`, running `ascp` on ESS I/O nodes is *not* supported. It is recommended to run `ascp` on dedicated nodes or on protocol nodes.

2.5 IBM Spectrum Scale configuration

IBM Spectrum Scale provides the source and the target file systems and directories. The IBM Aspera Sync file transfer reads the files on the source and writes the files to the target.

The IBM Spectrum Scale storage configuration is designed to support the required throughput. In addition, other jobs (user I/O, backup, tiering, and so on) that run at the same time as the replication must be taken into account for the disk sizing because these jobs might cause more I/O.

When `async` is used without the IBM Spectrum Scale policy engine, high metadata I/O results on the source file system because it gathers the file stat information. Consider the use of fast disk storage (SSD or Flash) for the file system metadata.

The following IBM Spectrum Scale configuration parameters can be adjusted (for more information, see [IBM Knowledge Center](#)):

- ▶ `maxFilesToCache`: Controls the number of files that are cached simultaneously. Because IBM Aspera Sync opens many files on the source for transfer, the value of this parameter must be large enough and aligned with the `async` option `--pending-max=N`, which defines the maximum number of files that are pending. This `async` option acts as a buffer to ensure that the number of transferred files does not exceed the maximum (default is 2000).
- ▶ `maxStatCache`: Controls the number of file attributes to cache in addition to what is cached in `pagepool`. When possible, set this parameter to the number of files that are stored in the source directory. If this amount cannot be determined, set it at least to the number of files that are added between replication cycles.

Note: For IBM Spectrum Scale on Linux, this option must be set to 0 when IBM Spectrum Scale version 5.0.1 or earlier is run.

- ▶ Pagepool: Amount of memory that is used to cache file data and metadata. Because the copy process is sequential, the source and target system can benefit from large pagepools.
- ▶ WorkerThreads: Controls the number of read and write threads. The number of worker threads must be increased, especially when multiple async or ascp threads are used or processes at the same time. The transfer threads with async can be controlled by using the --transfer-threads option.

To replicate the file set structure that is used within the source directory to the target, the file sets must be created manually in the target file system. IBM Aspera Sync has no awareness for file sets; instead, it considers these directories.

If a directory on the target is a manually created file set, IBM Aspera Sync copies the files into this file set.



Using IBM Aspera Sync with IBM Spectrum Scale

In this chapter, we describe how to use `async` and `ascp` without and with the IBM Spectrum Scale policy engine.

This chapter includes the following topics:

- ▶ 3.1, “IBM Spectrum Scale policy engine considerations with IBM Aspera” on page 28
- ▶ 3.2, “Async without policy engine” on page 29
- ▶ 3.3, “Ascp without policy engine” on page 30
- ▶ 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30
- ▶ 3.5, “Integration with Hierarchical Storage Management” on page 45
- ▶ 3.6, “Integration with immutable files” on page 51
- ▶ 3.7, “Integration with IBM Spectrum Scale Snapshots” on page 54

Note: The programs (scripts) and policies that are presented are intended for educational purposes and might not work in real-world deployments. For more information about a set of exemplary scripts and policies, see this [GitHub web page](#).

3.1 IBM Spectrum Scale policy engine considerations with IBM Aspera

IBM Aspera Sync can be used with or without the IBM Spectrum Scale policy engine. For more information, see 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30.

The IBM Spectrum Scale policy engine provides a fast way to identify files according to certain criteria; for example, files that were created or modified within a certain period.

The policy engine can be programmed with rules that express the criteria for file identification. The file (and directory) names that are identified according to the criteria are provided in file lists, which can be further processed by IBM Aspera Sync. The identification of files to be transferred is performed by the IBM Spectrum Scale policy engine while IBM Aspera Sync transfers the files according to overwrite and deletion policies. For more information, see 3.4.1, “IBM Spectrum Scale policy engine” on page 31.

The IBM Spectrum Scale policy engine as a method for fast file identification is used when many files are stored in the source directory. With many files in the source directory, the identification of files subject for transfer can take a long time when the standard async process is used because async crawls through the source directory to identify files based on their attributes (stat information).

Crawling through a large file system increases the workload on the metadata disks of the IBM Spectrum Scale file system and can take longer periods. This issue can cause negative side effects in the IBM Spectrum Scale cluster. The async session also can take a long time, which causes subsequent async sessions to overlap with previous sessions. This issue can be prevented by integrating async with the IBM Spectrum Scale policy engine. For more information, see 3.4.2, “Integration with async” on page 37.

Ascp does not identify files; instead, it gets file and directory names and patterns as input parameters. The file and directory names to be transferred by ascp can be identified by the IBM Spectrum Scale policy engine.

Because the policy engine creates lists of file names that match the selection criteria, these lists of file names can be passed into the ascp tool. Therefore, integrating ascp with the IBM Spectrum Scale policy engine makes sense because the policy engine can efficiently identify new and modified files and pass these file names in lists to the ascp program. For more information, see 3.4.3, “Integration with ascp” on page 41).

IBM Aspera Sync also can use snapshots as the source for the transfer (see 3.7, “Integration with IBM Spectrum Scale Snapshots” on page 54). No native integration with hierarchical storage management tools (HSM) is available in IBM Spectrum Scale. However, if IBM Aspera Sync is integrated with the IBM Spectrum Scale policy engine, files that are migrated can be excluded from the file transfer (see 3.5, “Integration with Hierarchical Storage Management” on page 45). IBM Aspera Sync can transfer immutability attributes of files, which is described in 3.6, “Integration with immutable files” on page 51).

3.2 Async without policy engine

Async can identify new, modified, and deleted files on the source and transfer these files to the target. This integration of async with IBM Spectrum Scale assures that the extended attributes, such as ACL, owner, user attributes, and immutability attributes, are preserved at the target.

To use async to replicate a source directory to a target directory, the async command can be scheduled through an operating system scheduler or an external scheduler. As described in 2.3, “How async works” on page 11, we provided several examples that address different requirements. An async summary example is shown in Example 3-1.

Example 3-1 Async summary example

```
# async -N sync_fs1 -d /fs1 -r user@remotehost:/target/fs1
--preserve-xattrs=native -i ~/.ssh/id_rsa --preserve-uid --preserve-gid
--preserve-time [-K push --overwrite=always --delete-delay]
```

When async is used for a specific pair of source and target directories, it is important to always use the same session name (parameter **-N**) because this parameter uses the existing Sync database (see 2.3.2, “Sync Database” on page 12).

When a different session name is used, a new Sync database is created and all files in the source and in the target directory must be inventoried in the new Sync database. Depending on the number of files in the source and target directory, this inventory process can take much time.

In addition, it is important to run an async session on one node of the source cluster. Distributing the workload of one async session to multiple nodes is not supported because only one node can obtain the lock for the Sync Database.

Example 3-1 uses the push mode (which is the default) where files are pushed from source to the target. The overwrite policy is set to always (which is the default) where files that were identified by the async process to be transferred overwrite the file instance on the target.

Files that were deleted on the source are also deleted on the target at the end of the async session, which is controlled by the `--delete-delay` parameter. By default, files that are deleted on the source are deleted on the target during the async session.

When async is not integrated with the IBM Spectrum Scale policy engine, it identifies the files that were created or modified in the traditional way. Therefore, it determines the status of each file and directory in the source directory to identify files that were created or modified since the last async session. Stating files and directories in large file system with millions of files can take a considerable amount of time. It also creates many small I/O for the metadata disks in the source IBM Spectrum Scale file system. If these two aspects affect the replication duration, it is recommended to explore the integration of async with the IBM Spectrum Scale policy engine.

Note: For large source directories with many files, it is recommended to integrate async with the IBM Spectrum Scale policy engine.

3.3 Ascp without policy engine

Ascp cannot identify files in the source directory that must be transferred. This identification normally is performed by async. Ascp is the file transfer tool that is used by async that is responsible to transfer files from the source to the target directory. The file and directory names to be transferred are provided by async.

Async features all of the optimizations to identify files that must be transferred and start ascp. However, ascp can be integrated with the IBM Spectrum Scale policy engine, whereby IBM Spectrum Scale identifies the files to be transferred and ascp transfers those files. For more information, see 3.4.3, “Integration with ascp” on page 41.

In some cases, it might be useful to use ascp independent of async or the IBM Spectrum Scale policy engine’ for example, in cases where large files must be transferred in a non-automated environment. When this transfer is performed, ascp does not update the Sync database that is managed by async. Therefore, certain information about files that were transferred bypasses the async controls. The use of ascp is shown in Example 3-2.

Example 3-2 Example of ascp command

```
# ascp -l 1G --preserve-xattrs=native -d -p --preserve-xattrs=native
--preserve-file-owner-gid --preserve-file-owner-uid -i ~/.ssh/id_rsa /fs1/*
user@remotehost:/target/fs1/ [--overwrite=diff -k 1 --delete-before-transfer]
```

As shown in Example 3-2, files from the source directory /fs1 are copied to the target directory /target/fs1. The IBM Spectrum Scale extended attributes, user-ID, and group-ID also are preserved. Files that are on the target are transferred and overwritten only if they have a different file size than on source (such as parameters --overwrite=diff -k 1, see 2.3.5, “Control overwrites” on page 14). The parameter --delete-before-transfer takes care to delete files in the target directory that were deleted in the source directory (see 2.4.4, “Control deletes on target” on page 20).

Ascp does not update the Sync Database with the file it copied. Therefore, a subsequent async run for the same files and directories must inventory all files that are copied by ascp and determine their checksum and state in the source and target directory. It might be useful to run ascp periodically (such as multiple times a day) to copy files from the source to the target. Async can be run less frequently (such as once a week on the weekend) to assure consistency in case ascp was not successful in copying files during the daily runs.

3.4 Integration with the IBM Spectrum Scale policy engine

The IBM Spectrum Scale policy engine provides a fast way to identify files according to certain criteria; for example, files that were created or modified within a certain period. The policy engine can be programmed with rules that express the criteria for file identification (see 3.4.1, “IBM Spectrum Scale policy engine” on page 31).

The file (and directory) names that are identified according to the criteria are provided in file lists that can be further processed by IBM Aspera Sync. Therefore, the IBM Spectrum Scale policy engine identifies the files that are to be transferred while IBM Aspera Sync transfers the files according to overwrite and deletion policies.

IBM Aspera Sync tools async and ascp can be integrated with the IBM Spectrum Scale policy engine. The difference between these tools is that async tracks the status of files and directories in the source and target directory that it transfers, ascp does not.

This difference comes with a price: `async` includes more overhead. However, `async` does not crawl through the file system to identify files subject for transfer; instead, it picks the file names that are provided by the policy engine in one or more file lists.

Two methods are available to integrate IBM Aspera Sync with the policy engine: the use of a stand-alone LIST policy and the use of a LIST policy with an external script. The policy rules for file identification are identical for both methods.

When a stand-alone LIST policy is used, two steps are required. In the first step, the IBM Spectrum Scale policy engine is started with a LIST policy to create lists of files names that match certain criteria of the policy.

In the second step, IBM Aspera Sync is started to transfer the file having their names included in the file lists. The advantage of this method is that the administrator has more control over the file lists and the execution of IBM Aspera Sync. For example, the administrator can adjust the file lists that are generated by the policy engine. They also can start the IBM Aspera Sync sessions with different parameters. The disadvantage of this method is that it requires two steps that must be orchestrated.

When LIST policy is used with an external script, only one step is required whereby the policy engine is started with a LIST policy to create lists of files names according to the criteria and start an external script. The external script obtains the list of files that are identified by the policy engine as input parameter and starts IBM Aspera Sync. The advantage of this method is that it requires one step. The disadvantage is that the administrator has less control over the file lists and the parameters that are used for the IBM Aspera Sync sessions.

Before we further explain the integration of IBM Aspera Sync with the IBM Spectrum Scale policy engine by using these two methods, provide an overview about the IBM Spectrum Scale policy engine.

3.4.1 IBM Spectrum Scale policy engine

The IBM Spectrum Scale policy engine allows the fast identification of files based on their attributes. The policy engine is represented by the `mmapplypolicy` command. For more information, see [IBM Knowledge Center](#).

This command takes a policy file as input and identifies the files based on the rules that are included in these policies. The policy file is a plain text file that is written in the IBM Spectrum Scale policy syntax. For more information, see [IBM Knowledge Center](#).

The rules define the selection criteria for the files based on file attribute and actions to be performed with the selected files. The actions that are performed by the policy engine in the context of the integration with IBM Aspera Sync are to provide a file list including the names of the selected files or to start IBM Aspera Sync with these file lists directly through an external script.

The policy engine is run on the source system. The basic syntax of the `mmapplypolicy` command is shown in Example 3-3.

Example 3-3 mmapplypolicy basic syntax

```
# mmapplypolicy /fsl -P policyfile -N asperanode -m 1 -B 1000 --single-instance  
-s localdir -g globaldir -I test|defer|yes [-f fileprefix]
```

The command features the following parameters:

/fs1	The source directory where files must be identified.
-P policyfile	The file that includes the rules for this policy.
-N asperanode	Specifies a single IBM Spectrum Scale node name where the IBM Aspera Sync tools are installed and running.
--single-instance	Specifies that only one instance of the policy engine can run. If another instance is running, this command stops.
-m 1	Indicates that only one thread began to process the selected files. In many cases, the use of one thread is sufficient on a single node.
-B 1000	Specifies the number of file names in one file list. In this case, it is 1,000 files per file list.
-s localdir	Specifies a local directory that is used to store temporary files that are created by the policy engine. Sufficient space must be available in this directory. The default directory is /tmp.
-g globaldir	Specifies a directory that is accessible to all cluster nodes. It can be in the file system that is processed by the policy engine or in a different file system. The default is specified by the IBM Spectrum Scale configuration parameter sharedTmpDir.
-I test defer yes	Specifies the mode for the policy run. Test means that the policy is tested for syntax. Defer indicates that the policy performs the file selection but does not start the action. Yes indicates that the policy performs the file selection and runs the action.
-f fileprefix	Specifies a prefix for an output file that is generated with a list policy when it is run in deferred mode.

The `policyfile` file is the policy that includes the rules for the file selection. The rule that is shown in Example 3-4 identifies and selects files that were created or modified (including file metadata modifications only) since the time stamp 2018-06-06 12:37:00.

Example 3-4 Rule identifies and selects files that were created or modified

```
RULE 'asperaRule' LIST 'files' WHERE  
( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR  
  CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") )
```

The policy engine identifies the file names that match the rules and creates file lists with these file names included. This file list includes files names and other metadata for each file in the following format:

```
48900 1741777473 0 -- /fs1/file1  
  
48901 7383947666 0 -- /fs1/file2
```

The file list that is shown in Example 3-4 includes two records with one per line. Each record includes five columns that are separated with blanks. The first three columns are IBM Spectrum Scale internal numbers that describe the file (`inodenum`, `inodegeneration`, and `snapid`). The file name is the fifth field in the file list. Each record starts in a new line.

Depending on the policy rules, the policy engine creates the file lists or starts an external script and passes the file name of the file lists to this program.

With this basic knowledge about IBM Spectrum Scale policies, we explain the concept of stand-alone LIST policies and LIST policies with an external script next. Then, we explain the integration of IBM Aspera Sync by using these two methods. For more information and guidance for using the policy engine, see the white paper [IBM Spectrum Scale Archiving Policies: An introduction to GPFS policies for file archiving with Spectrum Archive Enterprise Edition](#).

Stand-alone LIST policies

With stand-alone LIST policies, the IBM Spectrum Scale policy engine is used to create a file list that includes file names that were identified according to the criteria of the policy. It stores this file list in a specified directory.

Example 3-5 shows a policy with three rules. The first rule in this example defines a macro that specifies directory and file names that are excluded from the identification process. The second rule defines that the policy engine creates a list of files. The third rule specifies the selection criteria of the files that were created or modified after a certain time stamp; in this example, `TIMESTAMP("2018-06-06 12:37:00")`:

Example 3-5 Policy with three rules

```
/* 1. Rule: macro to define the files and directories to be excluded */
define(
    exclude_list,
    (PATH_NAME LIKE '%/.SpaceMan/%'
    OR PATH_NAME LIKE '%/.ctdb/%'
    OR PATH_NAME LIKE '%/.private-asp/%'
    OR PATH_NAME LIKE '%/.mmSharedTmpDir/%'
    OR PATH_NAME LIKE '%/.snapshots/%'
    OR NAME LIKE 'user.quota%'
    OR NAME LIKE 'fileset.quota%'
    OR NAME LIKE 'group.quota%')
)

/* 2. Rule: external list rule with no interface script */
RULE EXTERNAL LIST 'modfiles' EXEC ''

/* 3. Rule: file selection rule */
RULE 'asperaRule' LIST 'modfiles' WHERE
( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
  CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") ) AND
  NOT (exclude_list)
```

The time stamp in the third rule (`TIMESTAMP("2018-06-06 12:37:00")`) is the date and time of the previous replication session. All files that were created or modified (including file attribute modification) are selected with this rule.

To run this policy, use the `mmapplypolicy` command in deferred mode, as shown in Example 3-6. For more information about the parameter that is used with this command, see 3.4.1, “IBM Spectrum Scale policy engine” on page 31.

Example 3-6 mmapplypolicy command in deferred mode

```
# mmapplypolicy /fsl -P policyfile -N asperanode -f policy --single-instance -I
defer [-s localdir -g globaldir]
```

For more information about the parameters that are used with the `mmapplypolicy` command, see 3.4.1, “IBM Spectrum Scale policy engine” on page 31. Parameters `-B` and `-m` are not required with stand-alone LIST policies.

The `mmapplypolicy` command that is shown in Example 3-6 creates a file list that is named `policy.list.modfiles` whereby the first part of the file name comes from the `-f` option, the second part is fixed, and the third part of this name comes from the EXTERNAL LIST name of rule 2. This file list includes files that were identified according to the criteria that is specified in rule 3. This file list features the following format:

```
48900 1741777473 0 -- /fs1/file1
```

One line is sued for each file that was identified by the policy engine. The format of the line that describes the one file contains five fields. The first three fields are IBM Spectrum Scale internal numbers (inodenum, inodegeneration, and snapid). The file name is the fifth field in the file list.

This file list can be adjusted and split into multiple file lists and fed into the IBM Aspera Sync tools. For more information, see 3.4.2, “Integration with async” on page 37, and 3.4.3, “Integration with ascp” on page 41.

LIST policies with external script

A LIST policy can also be programmed in a way that it directly starts an external script that processes the file lists that are generated by the policy engine. During the processing in the external script, the file lists are adjusted and passed to the IBM Aspera Sync tools.

A policy with three rules is shown in Example 3-7. The first and the third rules are identical to the stand-alone LIST policy. The second rule includes the name of the external script (`/usr/local/bin/myscript.sh`) that must be programmed to process the file lists. The third rule specifies the selection criteria for files that were created or modified after a specific time stamp: (TIMESTAMP(“2018-06-06 12:37:00”).

Example 3-7 Policy with three rules

```
/* 1. Rule: macro to define the files and directories to be excluded */
define(
    exclude_list,
    (PATH_NAME LIKE '%/.SpaceMan/%'
    OR PATH_NAME LIKE '%/.ctdb/%'
    OR PATH_NAME LIKE '%/.private-asp/%'
    OR PATH_NAME LIKE '%/.mmSharedTmpDir/%'
    OR PATH_NAME LIKE '%/.snapshots/%'
    OR NAME LIKE 'user.quota%'
    OR NAME LIKE 'fileset.quota%'
    OR NAME LIKE 'group.quota%')
)

/* 2. Rule: external list rule with no interface script */
RULE EXTERNAL LIST 'modfiles' EXEC '/usr/local/bin/myscript.sh'

/* 3. Rule: file selection rule */
RULE 'asperaRule' LIST 'modfiles' WHERE
( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
  CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") ) AND
  NOT (exclude_list)
```

The time stamp in the third rule (TIMESTAMP("2018-06-06 12:37:00)) is the date and time of the previous replication session. All files that were created or modified (including file attribute modification) are selected with this rule.

To run this policy, use the command that is shown in Example 3-8. For more information about the parameter that is used with this command, see 3.4.1, "IBM Spectrum Scale policy engine" on page 31.

Example 3-8 Run the mmapplypolicy command

```
# mmapplypolicy /fs1 -P policyfile -N asperanode -m 1 -B 1000 --single-instance  
[-s localdir -g globaldir]
```

The use of the **mmapplypolicy** command starts the policy engine, identifies the file according to the third rule, stores the selected file names in file lists with up to 1000 entries (as specified by using the **-m 1000** parameter in the **mmapplypolicy** command), and starts the external script (as specified in rule 2, `/usr/local/bin/myscript.sh`).

The policy engine starts one instance of the external script (as specified by using the **-m 1** parameter in the **mmapplypolicy** command) and passes one file list to this instance. If this instance of the external script finished processing and more file lists with up to 1000 entries exist, another instance is started with another file list. The external script is started on the node that is specified by parameter **-N** of the **mmapplypolicy** command).

The number of entries per file list (parameter **-B 1000** in the **mmapplypolicy** command) can be adjusted, depending on the size of the files named in the file list. For larger files, the value for parameter **-B** can be smaller than 1000; for small files, it can be greater.

The number of instances of the external script that is started in parallel by the policy engine (by using the **-m 1** parameter in the **mmapplypolicy** command) can also be adjusted. When **async** is used in the external script, the recommended value for this parameter is 1 because **async** can start multiple transfer threads. When **ascp** is used, the value of this parameter can be greater than 1.

The node name that is specified by using the **-N** parameter of the **mmapplypolicy** command must be a name of node (IBM Spectrum Scale node name) that includes the IBM Aspera Sync tools that are installed and configured and a network connection to the target cluster.

When **async** is used, it is recommended to specify one node name for each **async** session name. It is *not* recommended to run **async** for a specific session name in parallel on multiple nodes. When **ascp** is used, multiple node names can be specified to allow **ascp** in parallel on multiple nodes. In this case, it is recommended to configure multiple target nodes as well, whereby each source node transfers file to a distinct target node.

The external script (named `/usr/local/bin/myscript.sh` in rule 2) now must process the file list. Before we describe how to process these file lists by using IBM Aspera Sync, we briefly explain how the external script works in general.

The external script receives two or more of the following parameters from the policy engine:

- ▶ Parameter 1: String that describes the operation, which is **TEST** and **LIST** in this example. The external script is first started by using **TEST** as the first parameter and secondly with **LIST** as the first parameter. The invocation with the first parameter set to **TEST** allows the script to perform some checks; for example, assure IBM Aspera Sync is installed.

- ▶ Parameter 2: Depends on parameter 1. If the first parameter is LIST, the second parameter is the name of the file list, including the identified file names. If the first parameter is TEST, the second parameter is the name of the file system.
- ▶ Parameter 3 (optional): Specified by using the OPTS clause in the second rule.

After the external script is started with the first parameter set to LIST, it obtains the name of the file list as second parameter. The file list includes the file names that were identified by the policy engine. The format of the file list is shown in the following example:

```
48900 1741777473 0 -- /fs1/file1
```

The three first numbers are IBM Spectrum Scale internal numbers (inodenum, inodegeneration, and snapid). The file name is the fifth field in the file list.

By using this file list, the external script can extract and process the file names. Example 3-9 shows some pseudocode in Korn shell script semantic for the implementation of the external script. Based on the string that describes the operation in the first parameter, the second parameter might be the name of the file system (TEST) or the name of the policy result file (LIST).

Example 3-9 External script that can extract and process the file names

```
#!/bin/ksh

# pseudocode example for the script /usr/local/bin/myscript.sh

# function to process the file list provide by the policy engine
function process
{
    #this function is invoked with one argument that is the name of the filelist
    listFile=$1

    #now processes the file list referenced by variable $fileList
}

##### MAIN #####

# assign parameters received from the policy engine
opCode="$1"
fName="$2"
opts="$3"

# check parameters
...

# based on op code perform action
rc=0
case $opCode in
    TEST )
        # $1 is the policy operation ($opCode)
        # $2 is the file system name ($fName)
        # $3 option given with the second rule ($opts)

        # check if the file system exists, return nonzero if the checks failed
        echo "TEST option received for directory $fName."
        if [[ ! -z "$fName" ]] then
            if [[ -d "$fName" ]] then
```



```

        echo "TEST directory $fName exists."
    else
        echo "WARNING: TEST directory $fName does not exists."
        rc=1
    fi
fi;;

LIST )
#$1 is the policy operation ($oCode)
#$2 is the policy file name ($fName)
#$3 option given with the second rule ($opts) - does not apply here
#check if the file list exists
echo "LIST option received, starting receiver task"
if [[ ! -z "fName" ]] then
    if [[ -a "fName" ]] then
        echo "LIST file name $fName exists."
    else
        echo "WARNING: LIST file name $fName does not exists."
        exit 1
    fi
fi

#process the file list with a generic function process()
echo "Processing file $filename"
process $fName
rc=$?;;

* )
echo "Unknow argument $1 received"
rc=1;;
esac

# exit the script with the $rc assigned above
exit $rc

```

The function process() that is shown in the pseudocode in Example 3-9 on page 36 must be implemented according to the needs. For more information about how to implement this function process() for async, see “Async with external script” on page 39; for ascp, “Ascp with external script” on page 43.

3.4.2 Integration with async

Async can transfer files from a file list, including the file names that are subject for transfer from source to target, as described in 2.3.7, “Using file lists” on page 15. Async can be started manually with a file list that is generated by the policy engine with the stand-alone LIST policy or automatically by the policy engine with an external script.

Important: When async is used with file lists that are provided by the policy engine, files that are deleted in the source directory are not deleted in the target directory. Also, files that were renamed in the source directory are transferred as new files to the target directory and the old file persists in the target directory. More measures must be taken to promote deletions and renames to the target, as described next.

Async with stand-alone LIST policies

To use async in association with stand-alone LIST policies, the policy engine must be started by using an appropriate policy to create a file list that includes the file names of the identified files. An example of the policy file and the invocation of the policy engine is described in “Stand-alone LIST policies” on page 33. As a result of this policy run, the policy engine produced a file list (`policy.list.modfiles`) that includes the path and file names of files that matched the criteria of the policy.

This file list that is provided by the policy engine must be adjusted before it can be passed to async by using the `--include-from=filelist` parameter. The file list must be adjusted because the format of the file list that is created by the policy engine does not match the format that is expected by async.

Async requires the path and file name relative to the source directory. The policy engine provides the complete path and file name that are relative to the file system root as the fifth column. Therefore, the file and path name must be extracted from the file list that is provided by the policy engine and adjusted to be relative to the source directory.

To adjust the file list that is created by the policy engine to match the format that is expected by async, the command that is shown in Example 3-10 can be used. The source directory is `/fs1`, the file list that is provided by the policy engine is named `policy.list.modfiles` and the file includes the names of the files to be transferred is named `async.transfer.list`.

Example 3-10 Adjust the file list created by the policy engine to match the format expected by async

```
# awk -v source_dir="/fs1" '{ path = "";for(i = 5; i<=NF;i++) { path = path (i>5? " ":"") $i };sub(source_dir"/", "", path); n=split(path, elems, /\//); for(i = 1; i<=n -1;i++) { partial = elems[1] "/"; for (j=2; j<=i; j++) { partial = partial elems[j] "/" } print partial } print path }' policy.list.modfiles | sort | uniq > async.transfer.list
```

Note: The adjustment of the file list that is provided by the policy engine that is shown Example 3-10 tolerates files with blanks in their name. However, it does not tolerate file names with multiple consecutive blanks in their name.

After the file list that is produced by the policy is adjusted to match the async format (file name `async.transfer.list` as shown in Example 3-10), it can be passed to async by using the parameter that is shown in the following example:

```
--include-from=async.transfer.list
```

The async command that uses the adjusted file list is shown in Example 3-11.

Example 3-11 async command that uses the adjusted file list

```
# async -N sync_fs1 -d /fs1 -r user@remotehost:/target/fs1 -l 1G -K push  
-i ~/.ssh/id_rsa --preserve-xattrs=native --preserve-uid --preserve-gid -t  
--preserve-access-time --include-from=async.transfer.list --exclude "*" --create-dir
```

Note: Consider splitting the file list if the number of file names that are contained in this list is too high to be processed by a single async process. This split requires starting multiple async processes sequentially. Alternatively, consider increasing the number of threads that is started by async according to the number of files.

Async now processes each file that is provided in the file list (`async.transfer.list`) according to its logic by determining the file state on source and target, comparing the checksum by using the Sync database, and eventually transferring the file.

To promote deleted and renamed files to the target, `async` must be run without the policy engine. This process can be done periodically (for example, once a week) in times where the workload in the source directory is low. To run `async` without the policy engine, the command that is shown in Example 3-12 can be used.

Example 3-12 Run `async` without the policy engine

```
# async -N sync_fs1 -d /fs1 -r user@remotehost:/target/fs1
--preserve-xattrs=native -i ~/.ssh/id_rsa --preserve-uid --preserve-gid --preserve-time
```

The session name (`sync_fs1` in this example) must match the session name that is used for the transfer of files from a file list to use the existing Sync database.

Async with external script

Async can also be started through the policy engine by using a LIST policy with an external script. A pseudo code example of the external script (`/usr/local/bin/myscript.sh`) is shown in Example 3-9 on page 36. Async is started within the function `process()` of this example.

The function `process()` first adjusts the file list that is provided by the policy engine. Then, it starts `async` with the appropriate command line options. Example 3-13 shows the pseudo code for the adjustment of the file list and the invocation of `async`.

Example 3-13 Pseudocode for the function `process()` that can be integrated within the external script

```
#!/bin/ksh

# pseudo code example for function process that invokes async with the file list
(belongs to script /usr/local/bin/myscript.sh)

#some global variables
sourceDir="/fs1"
sessionName="sync_fs1"
sshUser="user"
sshKeyPath=~/.ssh/id_rsa"
sshHost="remotehost"
remoteDir="/target/fs1"
transferRate="1G"

# function to transfer the files in the file list
function process
{
    #first argument of this function is the name of the file list
    listFile=$1

    #extract file names from $listFile and store it in $listFileAsp
    listFileAsp=$1".aspera"
    awk -v source_dir=$sourceDir '{ path = "";for(i = 5; i<=NF;i++) { path = path
(i>5? " ":"") $i };sub(source_dir"/", "", path); n=split(path, elems, /\//); for(i
= 1; i<=n -1;i++) { partial = elems[1] "/" ; for (j=2; j<=i; j++) { partial =
partial elems[j] "/" } print partial } print path }' $listFile | sort | uniq >
$listFileAsp
```

```

#invoke async with the file list
async -N $sessionName -d $sourceDir -r $sshUser@$sshHost:$remoteDir
-l $transferRate -K push -i $sshKeyPath --preserve-xattrs=native -u -j -t
--preserve-access-time --include-from=$listFileAsp --exclude "*" --create-dir

#determine return code and exit
rc=$?
rm -f $listFileAsp
return $rc
}

```

The function `process()` that is shown in Example 3-13 on page 39 can be integrated into the sample external script `/usr/local/bin/myscript.sh` as described in “LIST policies with external script” on page 34.

Note: The extraction of the file names that is provided by the policy engine tolerates files with blanks in their name. However, it does not tolerate file names with multiple consecutive blanks in their name.

To run `async` that is started by an external script, a policy must be provided. This policy is run by using the `mmapplypolicy` command, as shown in Example 3-14.

Example 3-14 mmapplypolicy command running policy

```
# mmapplypolicy /fs1 -P policyfile -N asperanode -m 1 -B 1000 --single-instance
[-s localdir -g globaldir]
```

Note: The `-N` parameter must specify only one node; the `-m` parameter must be set to 1.

The use of the `mmapplypolicy` command starts the IBM Spectrum Scale policy engine. The policy engine identifies files based on their modification time, stores the file names in file lists, and starts the external script with one file list at a time.

The external script that implemented the function `process()` that is shown in Example 3-13 adjusts the file list and provides it to `async`. `Async` performs further checks on the files that are provided in the list and transfers the files when required.

After the external script processes the file list, it ends. If more file lists are generated by the policy engine, the external script is started again with a new file list and processes this until all file lists are processed.

With this implementation, files are identified by the IBM Spectrum Scale policy and not by `async`. If many files are stored in the source directory, the policy engine is faster than `async`.

`Async` processes these files that are identified by the policy engine by determining the file attributes, calculating the checksum, and comparing the file attributes and the checksum with the file that is stored on the target before transferring the file. It also creates and maintains a Sync database on the source and the target where it keeps track of the file attributes, checksum, and replication states. Therefore, it is important to keep the name of the session identical for a replication relation.

To promote deleted and renamed files to the target, `async` must be run without the policy engine. This task can be done periodically (for example, once a week) in times where the workload in the source directory is low. To run `async` without the policy engine, run the command that is shown in Example 3-15 on page 41.

Example 3-15 Run async without the policy engine

```
# async -N sync_fs1 -d /fs1 -r user@remotehost:/target/fs1  
--preserve-xattrs=native -i ~/.ssh/id_rsa --preserve-uid --preserve-gid --preserve-time
```

The session name (sync_fs1 in this example) must match the session name that is used in the function process() that starts the **async** command with the parameter **-N \$sessionName** denoting the session name.

3.4.3 Integration with ascp

Alternatively, the files that are identified by the IBM Spectrum Scale policy engine can be transferred by using ascp. As explained in 2.4, “How ascp works” on page 17, ascp is the backend of async and transfers the file from the source directory to the target directory that is under consideration for overwrite and delete policies.

It does *not* create and maintain a Sync database. For this purpose, ascp can transfer files from a file list, including the file names to transfer from source to target, as explained in 2.4.6, “Using file lists and file pair lists” on page 21.

The tool ascp can be started manually by using a file list that is generated by the policy engine with the stand-alone LIST policy (see , “Ascp with stand-alone LIST policies” on page 41) or automatically by the policy engine with an external script (see , “Ascp with external script” on page 43).

Important: When ascp is used with file lists that are provided by the policy engine, files that were deleted in the source directory are not deleted in the target directory. Also, files that were renamed in the source directory are transferred as new files to the target directory and the old file persists in the target directory. More measures must be taken to promote deletions and renames for files on the source to the target. Some of these measures are described next.

Ascp with stand-alone LIST policies

To use ascp in association with stand-alone LIST policies, the policy engine must be started by using an appropriate policy to create a file list, including the file names of the identified files. An example of the policy file and the invocation of the policy engine is described in “Stand-alone LIST policies” on page 33. As a result of this process, the policy engine produces a file list (policy.list.modfiles) that includes the path and file names of files that matched the criteria.

This file list must be adjusted before it can be passed to ascp by using the `--file-list=filelist` parameter. The file list must be adjusted because the format of the file list that is created by the policy engine does not match the format expected by ascp (by using file lists and file pair lists).

Ascp requires the fully qualified path and file name that is the fifth field within the file list that is provided by the policy engine. Therefore, the file and path name must be extracted from the file list that is provided by the policy engine.

To adjust the file list that was created by the policy engine to match the format that is expected by ascp, the command that is shown Example 3-16 on page 42 can be used. The use of this command extracts the full path and file names from the file list that is provided by the policy engine (policy.list.modfiles) and stores the resulting path and file names in file ascp.transfer.list.

Example 3-16 Extracts the full path and file names from the file list provided by the policy engine

```
# awk -F '[ ]' '{ for(i=7; i<=NF; i++) printf "%s", $i (i==NF?ORS:OFS) }'  
policy.list.modfiles > ascp.transfer.list
```

Note: The adjustment of the file list as shown in Example 3-16 is based on the standard output of the policy engine and tolerates blanks within the path and file names. However, it might no longer work if the output of the policy engine includes more fields; for example, when the SHOW statement is used within the policy rules.

After the file list that is produced by the policy is adjusted to match the ascp format (file name `ascp.transfer.list` as shown in Example 3-16), it can be passed to ascp by using the `--file-list=ascp.transfer.list` parameter, as shown in Example 3-17. In this example, files are transferred from the source directory that is named `/fs1` to the target directory that is named `/target/fs1`.

Example 3-17 Example of files transferred from the source directory to target directory

```
# ascp -l 1G -d --file-list=ascp.transfer.list --overwrite=diff -k 2 --mode=send  
--user=user --host=remotehost -i ~/.ssh/id_rsa -p  
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-gid  
--src-base=/fs1 /target/fs1
```

Note: Consider splitting the file list if the number of file names that is in this list is too high to be processed by a single ascp process. This task requires starting multiple ascp processes sequentially.

Ascp copies each file that is provided in the file list (`ascp.transfer.list`) from the source directory (`/fs1`) to the target directory (`/target/fs1`) that is considering the overwrite policies. It does *not* create and maintain a Sync database.

When files are deleted in the source directory, these files are not automatically deleted in the target directory. Likewise, when a file is renamed on the source, two copies are on the target: the file with old name and the file with the new name. The file with the old name on the target is redundant and can be deleted.

To promote the deletion of files to the target directory, ascp must be started without any file list and by using the ascp parameter `--delete-before-transfer`. The scope of the file transfer must include the entire source directory. This process causes all files on the source to be subject for transfer to the target.

To control whether files that are identical on source and target are transferred, use the ascp parameter `--overwrite` (for more information, see 2.4.3, “Control overwrites” on page 19). You can run ascp this way periodically (for example, once a week) and in times where the workload in the source directory is low. To run ascp for managing deleted and renamed files, the command that is shown in Example 3-18 can be used.

Example 3-18 Run ascp for managing deleted and renamed files

```
# ascp -l 1G -d -p --delete-before-transfer --overwrite=never  
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid -i  
~/.ssh/id_rsa /fs1 user@remotehost:/target/fs1
```

The use of this command deletes files on the target that were deleted or renamed on the source. Because the parameter `--overwrite=never` is used in Example 3-18 on page 42, files that are on the source and target are not copied. Files that are on the source and not on target are copied. Different overwrite policies can be used to control the processing in the target directory. For more information, see 2.4.3, “Control overwrites” on page 19.

Ascp with external script

Ascp can also be started by using the policy engine that uses a LIST policy with an external script. The name of the file list that is generated by the policy engine is passed to the external script (in this example, the external script is named `/usr/local/bin/myscript.sh`). This external script starts `ascp` with an adjusted file list. The invocation of `ascp` can be implemented in the function `process()` as shown in Example 3-18. For more information, see the adjustments that were done for `async` in “Async with external script” on page 39.

Before `ascp` can be started, the file list must be adjusted because the format of the file list that is provided by the policy engine does not match the format that is expected by `ascp`. For more information about this format adjustment, see “Ascp with stand-alone LIST policies” on page 41.

The adjusted file list is provided to `ascp` with the parameter `--file-list=filelist` (see 2.4.6, “Using file lists and file pair lists” on page 21). The pseudo code example that is shown in Example 3-19 shows how to start `ascp` with a file list that is produced by the policy engine.

Example 3-19 Starting ascp with a file list produced by the policy engine

```
#!/bin/ksh

# pseudo code example for function process that invokes ascp with the file list
(belongs to script /usr/local/bin/myscript.sh)

#some global variables
sourceDir="/fs1"
sessionName="sync_fs1"
sshUser="user"
sshKeyPath=~/.ssh/id_rsa"
sshHost="remotehost"
remoteDir="/target/fs1"
transferRate="1G"

# function to transfer the files in the file list
function process
{
    #first argument of this function is the name of the file list
    listFile=$1

    #extract file names from $listFile and store it in $listFileAsp
    listFileAsp=$1".aspera"
    awk -F '[ ]' '{ for(i=7; i<=NF; i++) printf "%s", $i (i==NF?ORS:OFS) }'
    $listFile > $listFileAsp

    #invoke ascp with the file list
    ascp -l $transferRate --file-list=$listFileAsp -d --overwrite=diff -k 2
    --mode=send --user=$sshUser --host=$sshHost -i $sshKeyPath -p
    --preserve-xattrs=native --preserve-file-owner-gid
    --preserve-file-owner-uid --src-base=$sourceDir $remoteDir
```

```
#determine return code and exit
rc=$?
rm -f $listFileAsp
return $rc
}
```

The function `process()` that is shown in Example 3-19 on page 43 can be with the sample external script `/usr/local/bin/myscript.sh`, as described in “LIST policies with external script” on page 34.

Note: The extractions of the file names from the file list that is provided by the policy engine tolerates files with blanks in their name. However, it relies on the standard output format of the policy engine. If more fields are in the file list, this extraction must be adjusted.

To run `ascp` that is started by an external script, a policy must be provided (for more information, see “LIST policies with external script” on page 34). This policy is run by using the `mmapplypolicy` command, as shown in Example 3-20.

Example 3-20 RUNNING the policy with the mmapplypolicy command

```
# mmapplypolicy /fs1 -P policyfile -N node1[,node2 ..] -m 2 -B 1000
--single-instance [-s localdir -g globaldir]
```

With `ascp` transferring files, the `-N` parameter can include more than one IBM Spectrum Scale node name where IBM Aspera Sync is installed. If multiple node names are specified, an instance of the external script is started with a separate file list.

The parameter `-m` can have a value of 1 or greater. With this parameter set to 2 as shown in Example 3-20, two instances of the external script are started on each node that is specified by using the `-N` parameter in parallel. Each instance of the external script processes one file list.

The use of the `mmapplypolicy` command starts the IBM Spectrum Scale policy engine. The policy engine identifies files based on their modification time, stores the file names in file lists, and starts the external script with one file list at a time.

The external script that implemented the function `process()` that is shown in Example 3-19 on page 43 adjusts the file list and provides it to `ascp`. `Ascp` transfers the files that are named in the file list. If one instance of the external script that is processing one file list finished, another instance of the external script is started with a new file list until all of the file lists are processed.

The integration of `ascp` with the IBM Spectrum Scale policy engine through an external script is an alternative to the use of `async`. With this integration, `async` is not required because the policy engine identifies the files to be transferred.

The use of the policy engine to identify files is fast. The use of `ascp` without `async` reduces the overhead because no further checking and metadata tracking (file attributes and checksums) is done for the files that are transferred.

To promote deleted and renamed files to the target, `ascp` must be run without the policy engine and by using the `--delete-before-transfer` parameter (see 2.4.4, “Control deletes on target” on page 20). This process can be done periodically (for example, once a week) in times where the workload in the source directory is low.

To run `ascp` without the policy engine to manage deleted and renamed files, the command that is shown in Example 3-21 can be used.

Example 3-21 Running ascp without the policy engine to manage deleted and renamed files

```
# ascp -l 1G -d -p --delete-before-transfer --overwrite=never
--preserve-xattrs=native --preserve-file-owner-gid --preserve-file-owner-uid -i
~/ .ssh/id_rsa /fs1 user@remotehost:/target/fs1
```

The use of this command deletes files on the target that were deleted or renamed on the source. Because the parameter `--overwrite=never` is used in the Example 3-21, files that are on the source and the target are not copied. Files that are on the source and not the target are copied. Different overwrite policies can be used to control the processing in the target directory (see 2.4.3, “Control overwrites” on page 19).

In summary, the use of `ascp` with the policy engine is simpler and more scalable than `async` because it can run across multiple nodes. However, renamed and deleted files must be handled by extra runs of `ascp` by using the parameters `--delete-before-transfer` and `--overwrite=never`, as shown in Example 3-21.

It also is recommended to periodically check the inventory of the source and the target to ensure that no files were missed during the replication process. The IBM Spectrum Scale policy engine can be used to generate lists of files for the inventory check of the source and the target file system.

3.5 Integration with Hierarchical Storage Management

IBM Spectrum Scale provides the capabilities to migrate files that are stored in a file system from one storage tier to another. A storage tier is a storage device with a certain characteristic, such as solid-state disk (SSD), Flash, disk, cloud storage, and tape.

To migrate files from disk to tape, more software components are required. These components can be IBM Spectrum Archive Enterprise Edition that migrates files from disk to tape that is formatted in the IBM Linear Tape File System™ format (LTFS). It also can be IBM Spectrum Protect for Space Management that migrates files from disk to an IBM Spectrum Protect server that can store the data on tape.

This migration capability from disk to tape is also called *hierarchical storage management* (HSM), which provides cost savings by moving data that must be retained and is no longer accessed to a cheaper storage tier, such as tape.

The basic concept of HSM is to migrate files from disk to tape while keeping migrated files transparently accessible in the file system. To identify files that must be migrated, IBM Spectrum Scale MIGRATE policies can be used with the policy engine. For more information, see [IBM Knowledge Center](#).

During the migration process, the HSM component copies the file to the destination tape by way of IBM Spectrum Archive or IBM Spectrum Protect and then, creates a stub of the migrated file. The stub is the inode of the file that makes the file visible in the file system. However, the content of the file is on tape at the destination.

When a migrated file is opened, the HSM component is started by IBM Spectrum Scale, the destination of the file is identified, and the file is copied back to disk. This process is called *recall*. After the file content is recalled, access is granted to the process that opened the file.

The recall from tape takes some time because the tape must be mounted and spooled. Therefore, it is not recommended to recall many files at once.

When async identifies files to be transferred, it first checks if the file was modified based on the file modification time stamp. If the file was modified, async opens and reads the file to determine the checksum and compares it with the modification time stamp and checksum on the target.

If the file is migrated, this operation (open and read) causes the file to be recalled. If this issue occurs with many files during an async session, many recalls are run at the same time, which can affect the file system performance and the transfer performance of the async session.

Many concurrent transparent recalls are also called *recall storms* and can cause hangs in the file system and excessively long async sessions. It can even result in a condition where a subsequent async session starts although the previous session is not finished. Recall storms must be prevented because of their adverse effects.

Important: In an environment with HSM and IBM Aspera Sync, it is recommended to prevent changes of the files after they are transferred by IBM Aspera Sync. That is, only files are stored in the source directory that are not changed again. Otherwise, lengthy tape operations are inevitable that can have negative effect on the cluster and IBM Aspera Sync performance. Also, it must be ensured that IBM Aspera Sync transfers files from the source to the target before files are migrated on the source.

In the context of the integration of IBM Aspera Sync with IBM Spectrum Scale, HSM that can be represented by the IBM Spectrum Archive Enterprise Edition or IBM Spectrum Protect for Space Management software can be installed and configured on the source and on the target directory.

Next, we describe both configurations and provide some best practices for the use of async. Although we focus on async, we also provide guidance for ascp.

3.5.1 Async with HSM on source

In this section, we focus on async with HSM on the source. We also provide some guidance for ascp as described in “Considerations for ascp” on page 49.

When async is used to transfer files from a source directory that is managed by HSM, it must be ensured that the files that are transferred were not migrated to tape before this process. Otherwise, the async opens and reads migrated files on the source directory to determine the checksum, which results in a recall. If this issue occurs with multiple files at the same time, recall storms are inevitable. For this reason, it must be ensured that files are replicated by async before they are migrated by HSM.

Important: Ensure that files are transferred by async before migrating files in the source directory.

To prevent file migration before replication by using async, files on the source directory can be migrated with a time delay of some days after they are created or modified. During this time delay, async can ensure that the file is replicated. A migration policy that migrates files 10 days after modification is shown in Example 3-22 on page 47. For more information, see *IBM Spectrum Scale Information Lifecycle Management Policies - Practical guide*, [WP102642](#).

Example 3-22 Migration policy that migrates files 10 days after modification

```
/* define exclude rule*/
RULE 'exclude' EXCLUDE WHERE (PATH_NAME LIKE '%/.SpaceMan/%' OR PATH_NAME LIKE
'%/.snapshots/%' OR PATH_NAME LIKE '%/.ltfsee/%' OR PATH_NAME LIKE
'%/.private-asp/%' OR PATH_NAME LIKE '%/.mmSharedTmpDir/%')

/* define macro for modification time */
define( mod_age,(DAYS(CURRENT_TIMESTAMP) - DAYS(MODIFICATION_TIME)) )
define( change_age,(DAYS(CURRENT_TIMESTAMP) - DAYS(CHANGE_TIME)) )

/* define file migration state */
define(is_migrated, (MISC_ATTRIBUTES LIKE '%V'))

/* define external pool on ltfs */
RULE EXTERNAL POOL 'ltfs' EXEC '/opt/ibm/ltfsee/bin/ltfsee' OPTS '-p pool1@lib1'

/* migrate files that have been modified 10 day ago or longer */
RULE 'MigLtfs' MIGRATE FROM POOL 'system' TO POOL 'ltfs' WHERE
( (mod_age > 10) OR (change_age > 10) ) AND NOT (is_migrated)
```

A better way to prevent recall storms that are caused by async is to integrate async with the policy engine (for more information, see 3.4.2, “Integration with async” on page 37). This integration allows you to exclude files that are migrated from being identified as candidates for the async session. Hence, migrated files do not become candidates for replication by async. An example of this policy is shown in Example 3-23.

Example 3-23 Exclude files that are migrated from being identified as candidates for the async session

```
/* define a macro for migrated files */
define(is_migrated, (MISC_ATTRIBUTES LIKE '%V'))

/* identify files that have been modified since the given time stamp and that are
not migrated */
RULE 'asperaRule' LIST 'files' WHERE
( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") ) AND
NOT (is_migrated)
```

As shown in Example 3-23, a macro is defined that specifies the state for migrated files. In the subsequent rule, files are not selected if they are migrated. This rule can be integrated into the policies for async as described in 3.4.2, “Integration with async” on page 37.

The condition that async skips migrated files that must be transferred must be resolved to have a consistent state between source and target. For this purpose, a stand-alone LIST policy can be implemented that identifies files that are migrated after the specified modification time stamp.

The modification time stamp matches the last async session. For more information about stand-alone LIST policies, see , “Stand-alone LIST policies” on page 33. Example 3-24 shows a policy that selects files that were migrated after a specified modification time.

Example 3-24 Policy that selects files that were migrated after a specified modification time

```
/* define macro to define the files and directories to be excluded */
define(
    exclude_list,
```

```

        (PATH_NAME LIKE '%/.SpaceMan/%'
        OR PATH_NAME LIKE '%/.ctdb/%'
        OR PATH_NAME LIKE '%/.private-asp/%'
        OR PATH_NAME LIKE '%/.mmSharedTmpDir/%'
        OR PATH_NAME LIKE '%/.snapshots/%'
        OR PATH_NAME LIKE '%/.lftsee/%'
        OR NAME LIKE 'user.quota%'
        OR NAME LIKE 'fileset.quota%'
        OR NAME LIKE 'group.quota%')
    )

/* define a macro for migrated files */
define(is_migrated, (MISC_ATTRIBUTES LIKE '%V'))

/* define external list rule with no interface script */
RULE EXTERNAL LIST 'modfiles' EXEC ''

/* identify files that have been modified since the given time stamp and that are
not migrated */
RULE 'asyncRule' LIST 'files' WHERE
( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
  CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") ) AND (is_migrated)

```

This policy can be run by using the `mmapplypolicy` command to create a list of files that match the criteria, as shown in Example 3-25.

Example 3-25 Using mmapplypolicy command to create a list of files matching the criteria

```
# mmapplypolicy /fs1 -P policyfile -N asperanode -f migrated --single-instance -I
defer [-s localdir -g globaldir]
```

The `migrated.list.modfiles` file list includes the file names of migrated files that are not yet transferred. This file list can be used to run bulk recalls by using the HSM component. Bulk recalls are optimized in a way that they sort the files by their location on tape and recall the files in this sequence. Therefore, the number of tape mounts is drastically reduced and the transfer speed is increased because all files that are stored on a specific tape are recalled in one operation. An example to start a bulk recall with IBM Spectrum Archive is shown in the following example:

```
# lftsee recall migrated.list.modfiles
```

When IBM Spectrum Protect for Space Management is used, the file list that is created by the policy engine must be adjusted to extract the path and file names. The command that is shown in Example 3-26 can be used.

Example 3-26 Adjust policy engine to extract the path and file names.

```
# awk -F '[ ]' '{ for(i=7; i<=NF; i++) printf "%s", $i (i==NF?ORS:OFS) }'
migrated.list.modfiles > recall.list
```

The recall can be run with IBM Spectrum Protect for Space Management. Specifying the file system path of the source directory perform a tape optimized recall, as shown in the following example:

```
# dsmrecall -filelist=recall.list [/fs1]
```

After these files are recalled, `async` can be run again through the policy engine to transfer these files to the target directory. After this process completes, the files that were recalled can be migrated again by using the migration command of IBM Spectrum Archive (`1tfsee migrate -s filelist`) or IBM Spectrum Protect for Space Management (`dsmmigrate -filelist=filelist`).

In summary, when the source directory is managed with HSM, it is important to ensure that files are replicated by `async` before they are migrated. This replication can be achieved by delaying the migration of files based on the modification time stamp.

Also, it can be ensured that files that are stored in the source directory are not changed again to prevent recalls and subsequent replication and migration. This process can be implemented by setting files to immutable on the source immediately after the files are stored. Immutable files cannot be changed. When integrating `async` with the IBM Spectrum Scale policy engine, it can be ensured that files that are migrated are not selected to be transferred by `async`.

Considerations for `ascp`

`Ascp` does not identify files that were created and modified in the source directory. Instead, `ascp` is given a file name or a file list that includes file names to be transferred from the source to the target. If a file that is migrated on source is processed by `ascp`, the file is recalled. If this issue happens to many files at the same time, recall storms are inevitable.

To prevent recalls storms, file names that are passed to `ascp` should not be migrated on source. The best way to achieve this condition is to integrate `ascp` with the IBM Spectrum Scale policy engine (see 3.4.3, “Integration with `ascp`” on page 41).

When `ascp` uses the IBM Spectrum Scale policy engine, the policy rules that are used to identify files that are subject for transfer can exclude files that are migrated on the source directory. A policy to exclude file that are migrated on source is described in 3.5.1, “Async with HSM on source” on page 46.

Another way to prevent recall storms from using `ascp` is to prevent overwrites by using the `--overwrite=never` parameter. However, this parameter causes inconsistencies between source and target directories because files that are modified on source are not transferred to the target.

Yet another approach is to set file to immutable on the source after the file is created. This process prevents modifications of the file on the source so it can be migrated at any time after it was set to immutable.

In summary, the best way to transfer files with `ascp` from a source directory that is managed by HSM is to integrate `ascp` with the policy engine and exclude files that were migrated from being identified by the policy engine.

3.5.2 Async with HSM on target

In this section, we focus on `async` with HSM on the target and provide some guidance for `ascp`, as described in “Considerations for `ascp`” on page 51.

With HSM on the target directory, files that were transferred by `async` to the target directory are eventually migrated by HSM. If a file is not modified on the source directory, this issue is not a problem.

However, if a file that was migrated on the target is modified in the source directory, it becomes a candidate for replication with the next async session. During the async session, the file on the source is compared to the file on the target and transferred because it was modified in the source directory. This transfer causes the file to be recalled in the target directory. If this issue happens with many files within an async session, it can cause recall storms on the target.

This situation cannot be prevented because the target migration policy is unaware of file modifications on source. Therefore, the best recommendation is to migrate files on the target only if they do not change on the source.

Note: Use only HSM that is configured for the target directory with async if files are not changed in the source directory.

If many files that are migrated on target are changed on the source, administrative measures can be implemented to start bulk recalls for these files on the target before the next async session. Bulk recalls are more efficient because they are tape optimized by sorting the files to be recalled by the tape ID and the location on tape.

The challenge is to detect the situation that files that are migrated on the target were modified on source. However, if this detection is possible, the files that are modified on the source must be identified regarding the path and file name on source. Then, the source path must be mapped to the corresponding target path. With the resulting file list, bulk recalls can be run on the target, as described in 3.5.1, “Async with HSM on source” on page 46.

One way to prevent modification for files in the source directory the IBM Spectrum Scale immutability feature can be used. For more information, see *IBM Spectrum Scale Immutability Introduction, Configuration Guidance, and Use Cases*, [REDP-5507](#).

With this feature, files on the source can be set to immutable. An immutable file cannot be modified, renamed, or deleted. Async can transfer immutable files from the source and retain the immutability setting on the target. The HSM component on the target can be configured to migrate only files that were set to immutable. This configuration assures that file do not change on the source, which prevents recalls on the target.

Example 3-27 shows an HSM MIGRATE policy on the target that migrates only files that are immutable and that were modified 10 days ago or longer.

Example 3-27 HSM MIGRATE policy

```
/* define exclude rule*/
RULE 'exclude' EXCLUDE WHERE (PATH_NAME LIKE '%/.SpaceMan/%' OR PATH_NAME LIKE
'%/.snapshots/%' OR PATH_NAME LIKE '%/.ltfsee/%' OR PATH_NAME LIKE
'%/.private-asp/%' OR PATH_NAME LIKE '%/.mmSharedTmpDir/%')

/* define macro for modification time */
define( mod_age, (DAYS(CURRENT_TIMESTAMP) - DAYS(MODIFICATION_TIME)) )
define( change_age, (DAYS(CURRENT_TIMESTAMP) - DAYS(CHANGE_TIME)) )

/* define macro for immutability */
define(is_immutable, (MISC_ATTRIBUTES LIKE '%X%'))

/* define file migration state */
define(is_migrated, (MISC_ATTRIBUTES LIKE '%V'))

/* define external pool on ltfs */
```

```
RULE EXTERNAL POOL 'ltfs' EXEC '/opt/ibm/ltfsee/bin/ltfsee' OPTS '-p pool1@lib1'

/* migrate files that have been modified 10 day ago or longer and that are
immutable*/
RULE 'MigLtfs' MIGRATE FROM POOL 'system' TO POOL 'ltfs' WHERE
( (mod_age > 10) OR (change_age > 10) ) AND
(is_immutable) AND NOT (is_migrated)
```

In summary, the best way to prevent negative effects that are caused by async with HSM configured on the target directory is to ensure that files are not modified in the source directory again after they are transferred to the target. Otherwise, modifying files on the source cause recalls (storms) on the target.

One practical approach is to set the file to immutable on the source, allow async to transfer only files that are set to immutable on the source, and migrate files that are immutable on the target. For more information about the use of IBM Aspera Sync with immutable files, see 3.6, “Integration with immutable files” on page 51.

Considerations for ascp

When ascp and migrating files are used on the target by using HSM, the same challenges as described for async (see 3.5.1, “Async with HSM on source” on page 46) exist. That is, if a file is modified on source, it is identified by the policy engine and transferred to the target. If the file on the target is migrated, it is recalled. If many files that are migrated on the target are sent from the source, recall storms occur.

One way to avoid recall storms by using ascp is to prevent overwrites on the target by using the `--overwrite=never` parameter.

However, this parameter causes inconsistencies between source and target directories because files that are modified on source are not transferred to the target.

Another approach is to set the immutable flag on the source after the file is created. This setting prevents modifications of the file on the source, so it can be migrated at any time after it is set to immutable.

In summary, when ascp is used with HSM configured on the target directory, files that were stored on the source are not changed again. This rule can be enforced by setting files to immutable. For more information, see *IBM Spectrum Scale Immutability Introduction, Configuration Guidance, and Use Cases*, [REDP-5507](#).

Another way is to not overwrite files on the target. However, this process can cause inconsistencies between the source and target directory contents.

3.6 Integration with immutable files

IBM Aspera Sync and ascp include the capabilities to preserve the extended attributes of files that are transferred from the source to the target. This feature also includes immutability attributes that are provided by IBM Spectrum Scale.

For more information, see *IBM Spectrum Scale Immutability Introduction, Configuration Guidance, and Use Cases*, [REDP-5507](#).

Example 3-28 shows some extended attributes of an immutable file including the immutability attributes (`immutable`, `appendOnly`, `indefiniteRetention`, and `expiration Time`).

Example 3-28 Shows some extended attributes of an immutable file

```
# mmlsattr -L file1
file name:          file1
metadata replication: 2 max 2
data replication:   2 max 2
immutable:          yes
appendOnly:         no
indefiniteRetention: no
expiration Time:    Wed Nov  9 10:00:00 2035
flags:
storage pool name:  system
fileset name:       worm
snapshot name:
creation time:      Tue Nov  8 21:52:07 2016
Misc attributes:    ARCHIVE READONLY
Encrypted:          no
```

The `immutable`, `appendOnly`, `indefiniteRetention`, and `expiration time` attributes that are shown in Example 3-28 on page 52 can be set on a per file basis within an immutable fileset. For more information, see *IBM Spectrum Scale Immutability Introduction, Configuration Guidance, and Use Cases*, [REDP-5507](#).

IBM Aspera Sync preserves immutability attributes for files that are transferred to the target directory. However, this process requires that the target directory features the same fileset structure as the source directory. In particular, the immutable filesets that were created on the source must be manually created on the target.

Certain immutability attributes can be changed by the owner of the immutable file on the source. For example, the `expiration time` can be increased (but not decreased for fileset configured in compliant mode), the `indefiniteRetention` flag can be set to `yes` or unset to `no`, and the `appendOnly` and `immutable` flags can be set from `no` to `yes`.

IBM Aspera Sync identifies and selects files for transfers, even if only file attributes changed (for more information, see 2.3, “How async works” on page 11). In this case, only the changed attributes are transferred and applied on the target file.

Ascp copies any file entirely that matches the file and directory pattern that is provided with the `ascp` command, including the updated file attributes. The following limitations apply with IBM Aspera Sync version 3.9 regarding the transfer of immutability attributes:

- ▶ If an immutable file was transferred to the target and later an extended attribute, such as `expiration time` or the `indefiniteRetention` flag for the file on the source are changed, `ascp` applies the changed attributes to the file on the target and present an error message. This error message can be prevented by setting the default file creation mask for `ascp` to `000`. For more information about setting the default file creation mask, see “File creation mask”.
- ▶ If an immutable file was transferred to the target, any subsequent transfer by using `ascp` always presents an error message. This issue occurs because `ascp` attempts to transfer the file entirely. The error message occurs because it is immutable on the target. If only the `expiration time` or the `indefiniteRetention` flag for the file on the source is changed, `ascp` applies this change to the target file; however, the error message persists.

- ▶ If a file with the `indefiniteRetention` flag set to `yes` was transferred to the target and later the `indefiniteRetention` flag is unset for the file on the source. Then, this change is not applied to the file on the target during subsequent transfers. At the time of this writing, this issue is under investigation.
- ▶ If a file with the `appendOnly` flag was transferred to the target and later this file is changed on the source, the next transfer of the changed file to the target fails. This issue occurs because `async` and `ascp` do not support append-only mode.
- ▶ Immutable filesets that are configured in compliant mode do not allow to unset the `immutable` flag for a file. However, normal filesets might allow this unset. IBM Aspera Sync (`async` and `ascp`) cannot unset the `immutability` flag on the target; therefore, the transfer fails.

Important: IBM Aspera Sync (`async` and `ascp`) does not support the transfer of changes of files where the `appendOnly` flag is set on the source. Also, `Async` does not unset the `indefiniteRetention` flag for files on the target when it was transferred from the source file.

When the expiration time or the `indefiniteRetention` flag of a file changes on the source that was transferred to the target, the new expiration time or `indefiniteRetention` flag is applied upon the next run, but it causes an error message.

File creation mask

When a file is created on the target, IBM Aspera Sync uses a default file creation mask. The default file creation mask is set to 644. Because immutable files on the source are read-only, the default mask on the target causes error messages when updating file attributes on the target, such as the expiration time or the `indefiniteRetention` flag.

To prevent this error message, the default file creation mask that is used by IBM Aspera Sync can be set to 000. By using this setting, the permissions of the file from the source file are applied on the target. To change the default file creation mask, the following command can be used on the target:

```
# asconfigurator -x "set_node_data;file_create_grant_mask,000"
```

Attribute changes based on time stamp

Changes of file attributes are reflected in change time of a file. The rule that is shown in Example 3-29 identifies files that were created, modified, or experienced changes to extended attribute after a specific time stamp.

Example 3-29 Rule identifies files with changes of attributes based on specific time stamp

```
RULE 'newfiles' LIST 'modfiles' WHERE
MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00")
```

The time stamp in this rule (`TIMESTAMP("2018-06-06 12:37:00")`) is the date and time of the previous run of the policy engine with `ascp` integration.

Note: Running `async` in bidirectional transfer mode might cause an error if the file names on the source and the target collide.

It is recommended to transfer files from the source to the target directory after the files are set to immutable and the expiration time and `indefiniteRetention` flag were set. In this way, it is ensured that the files are transferred once and do not change again.

3.7 Integration with IBM Spectrum Scale Snapshots

IBM Aspera Sync (async and ascp) can also replicate files from a snapshot that was created in the source file system or fileset that includes the source directory. The main advantage of the use of a snapshot as source for the replication is that files that are accessed or locked in the source directory can be replicated without access conflicts. Therefore, accessed or locked files do not affect the replication session.

In addition, the application can create an application consistent snapshot for applications that require a consistent recovery point, such as databases. This snapshot can be used as source for the replication and the data that was replicated to the target is also consistent from an application perspective.

Async does not create snapshots. The snapshot must be created manually by using the following IBM Spectrum Scale command:

```
# mmcrsnapshot fsname snapshot-name [-j fsetname]
```

The first parameter (fsname) is the name of the file system that is hosting the source directory. The second parameter (snapshot-name) is the name of the snapshot. The third parameter (-j fsetname) is optional and specifies the name of an independent fileset. When this parameter is used, the snapshot is created for the fileset only and is within this fileset.

The snapshot is created in a directory that is named `.snapshot/snapshot-name` that is relative to the file system root. For example, if the file system root is in directory `/fs1` and the snapshot's name is `sync_fs1_snap`, the snapshot directory is named `/fs1/.snapshots/sync_fs1_snap`. If a snapshot is created for an independent fileset, the `.snapshot` directory is relative to the fileset path.

Important: The name of the snapshot for a replication relation between a source and target directory must always be the same.

When snapshots are used, the source directory for the async session is the associated snapshot directory. The source directory name must always be the same for a specific replication relation and async session name.

The name of the snapshot is part of the snapshot directory and is part of the source directory to be replicated to the target directory. Therefore, the name of the snapshot must always be the same for a specific replication relation. This configuration can be achieved by creating the snapshot, running async, and upon completion of the async session, deleting the snapshot.

An alternative method allows creating snapshots with different names for subsequent async sessions with the same session name and the use of symbolic links to link the snapshot directory with a different name to a snapshot directory with a stable name. However, this method is not preferred.

The directory structure within the snapshot directory is preserved relative to the file system root. For example, assume the file system root `/fs1` contains the following files and directories:

```
/fs1/file1  
/fs1/dir1/file2  
/fs1/dir1/dir2/file3  
/fs1/dir3/file4
```

After a snapshot of the file system /fs1 with the name sync_fs1_snap is created, the snapshot directory /fs1/.snapshots/sync_fs1_snap contains the following files and directories:

```
file1
dir1/file2
dir1/dir2/file3
dir3/file4
```

Important: The path name for the Sync database must be explicitly given with the async command.

Snapshot directories are read-only. By default, async creates the Sync database within the source directory (for more information, see 2.3.2, “Sync Database” on page 12). Because the source directory is the snapshot directory that is read-only, the path for the local Sync database must be set to a path within the source directory. The following async parameter can be used for this purpose:

```
--local-db-dir=lbdbir
```

Note: The target directory cannot be a snapshot directory.

The target cannot be a snapshot directory. To create snapshots on the target after successfully completing the async session, the snapshot must be created manually by using the IBM Spectrum Scale `mmcrsnapshot` command.

To run async by using a snapshot of the source directory, the snapshot must be created first. Assume that the file system name is fs1 and the path of the file system root is /fs1, as shown in the following example:

```
# mmcrsnapshot fs1 sync_fs1_snap
```

The snapshot directory name that must be used as source directory name is /fs1/.snapshots/sync_fs1_snap.

To run async from the snapshot of the source directory, use the command that is shown in Example 3-30.

Example 3-30 Running async from the snapshot of the source directory

```
# async -N sync_fs1 --local-db-dir=/fs1 -i ~/.ssh/id_rsa -d
/fs1/.snapshots/sync_fs1_snap -r user@remotehost:/target/fs1
--preserve-xattrs=native -u -j -t -l 1G --preserve-access-time [--create-dir]
```

The session name that is provided with parameter -N and the source directory name that is on the snapshot and provided with the parameter -d and must remain the same for all subsequent async sessions. Therefore, after the transfer session is complete, the snapshot must be deleted. Before the next transfer session, a new snapshot must be created with the same name.

If files were renamed and deleted in the source directory, the next async session from a new snapshot propagates these changes to the target directory.

After the async session completes, delete the snapshot so it can be re-created immediately before the next async session starts, as shown in the following example:

```
# mmdelnsnapshot fs1 sync_fs1_snap
```

Note: when using snapshot async cannot be run in bidirectional transfer mode.

In summary, async can be integrated with IBM Spectrum Scale snapshots by using snapshot directories as the source directory for a transfer session. The snapshot must be created manually (or within a script) on the source before the transfer session and deleted after the transfer session. For subsequent transfer sessions with the same session name (same transfer relation), a snapshot with the same name must be created. Snapshots can also be manually created on the target after the async session completes.

3.7.1 Considerations with ascp and snapshots

Ascp can also be integrated with IBM Spectrum Scale snapshots by using the snapshot directory as the source directory for the transfer. It works like with async, but the name of the snapshot is not important because ascp does not keep track of directory and file names in a Sync database.

To transfer files from a snapshot of the source directory by using ascp, the snapshot must be created first by using the following command:

```
# mmcrsnapshot fs1 ascp_snap
```

If the source directory is /fs1, the files that are in the source directory can be found in the /fs1/.snapshots/ascp_snap snapshot directory.

The command that is shown in Example 3-31 can be used to transfer files from a snapshot directory on the source to the target directory.

Example 3-31 Transferring files from a snapshot directory on the source to the target directory

```
# ascp -l 1G -d -p --delete-before-transfer --preserve-xattrs=native  
--preserve-file-owner-gid --preserve-file-owner-uid -i ~/.ssh/id_rsa  
/fs1/.snapshots/ascp_snap/ user@remotehost:/target/fs1/
```

The parameter assures that files that were renamed or deleted in the source directory since the last snapshot are renamed and deleted in the target directory.

After the file transfer by using the ascp command that is shown in Example 3-31 successfully completes, the snapshot can be deleted by using the following command:

```
# mmdelnsnapshot fs1 ascp_snap
```

Note: The snapshot can also be kept as a recovery point for the source directory.

When keeping the snapshot of the source directory, the new snapshot must have a new name the next time an ascp transfer from a snapshot is performed. Alternatively, the existing snapshot can be deleted and a new snapshot with the same name can be created.

After the successful file transfer with ascp, a snapshot can also be created on the target directory; however, this process must be done manually by using the following command:

```
# mmcrsnapshot target ascp_target_snap
```

3.7.2 Using the policy engine with snapshots

The IBM Spectrum Scale policy engine can be used to identify new and modified files in a snapshot. For more information about the use of `async` and `ascp` with the IBM Spectrum Scale, see 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30.

To identify files in a snapshot, the policy file must be adjusted to remove the `.snapshot` directory from the exclude list (see Example 3-32).

Example 3-32 Rules to adjust policy file to remove snapshot directory from exclude list

```
/* 1. Rule: macro to define the files and directories to be excluded */
define(
    exclude_list,
    (PATH_NAME LIKE '%/.SpaceMan/%'
    OR PATH_NAME LIKE '%/.ctdb/%'
    OR PATH_NAME LIKE '%/.private-asp/%'
    OR PATH_NAME LIKE '%/.mmSharedTmpDir/%'
    OR PATH_NAME LIKE '%/.private-asp/%'
    OR NAME LIKE 'user.quota%'
    OR NAME LIKE 'fileset.quota%'
    OR NAME LIKE 'group.quota%')
)

/* 2. Rule: external list rule with no interface script */
RULE EXTERNAL LIST 'modfiles' EXEC ''

/* 3. Rule: file selection rule */
RULE 'asperaRule' LIST 'modfiles' WHERE
( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
  CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") )
```

The first rule defines the file and directory names to be excluded. The snapshot directory (`.snapshots`) was removed from the exclude list. The second rule defines the external list without an external script. The third rule defines the identification criteria for files that is based on the modification time.

The `mmapplypolicy` command is used to run the policy. To scan the snapshot, the snapshot path (including the source directory) must be given to this command, as shown in Example 3-33.

Example 3-33 Use the `mmapplypolicy` command

```
# mmapplypolicy /fs1/.snapshots/ -P policyfile -N asperanode -f policy
--single-instance -I defer [-s localdir -g globaldir]
```

The use of this command runs the policy engine and creates a file list, including the file names that match the third rule of the policy. This file list must be adjusted and can be processed by `async` (see 3.4.2, “Integration with `async`” on page 37) or `ascp` (see 3.4.3, “Integration with `ascp`” on page 41). When `ascp` is used, the `--src-base` parameter must reflect the snapshot directory (in this example, `/fs1/.snapshots`).



Summary

In this publication, we presented different options to run and integrate IBM Aspera async by using IBM Spectrum Scale.

In this chapter, we provide some quick guidance for the use of these options and highlight some use cases.

This chapter includes the following topics:

- ▶ 4.1, “Async or ascp standalone” on page 60
- ▶ 4.2, “Async or ascp integrated with the policy engine” on page 60
- ▶ 4.3, “Stand-alone LIST policies or external policy script” on page 60
- ▶ 4.4, “Use of snapshots” on page 61
- ▶ 4.5, “Use cases” on page 61

4.1 Async or ascp standalone

When IBM Aspera Sync is used without the integration with the IBM Spectrum Scale policy engine, it is highly recommended to use `async`. `Ascp` can be used for testing purposes or to transfer specific files as needed, but note the caveats that were discussed in 3.3, “Ascp without policy engine” on page 30.

If the duration to identify the files to be transferred becomes too long by using `async`, it is recommended to integrate IBM Aspera Sync with the IBM Spectrum Scale policy engine.

4.2 Async or ascp integrated with the policy engine

When IBM Aspera Sync is integrated in the IBM Spectrum Scale policy engine, `async` or `ascp` can be used in the backend to transfer the files that are identified by the policy engine. We recommend the use of `async`.

The advantage of the use of `async` is that it tracks transferred files. This ability is the foundation to run full synchronizations without copying files that were transferred previously.

Running full synchronizations of the source and target directory assures that all files are transferred as a precaution against files were being transferred. This issue can occur if the network connection failed during a transfer or if the policy rules were ineffective. Another advantage is that `async` provides more control for file transfer sessions and threads.

The use of `ascp` independent of `async` integrated with the IBM Spectrum Scale policy engine has less overhead because it does not maintain the Sync Database. This issue is also a disadvantage because `ascp` copies a file, even if the source and the target directory hold the same version of the file. Running full synchronizations by using `async` copies all files again because no Sync Database is available for files that are copied with `ascp`.

4.3 Stand-alone LIST policies or external policy script

As described in 3.4.1, “IBM Spectrum Scale policy engine” on page 31, the following options are available when `async` or `ascp` is integrated with the IBM Spectrum Scale policy engine:

- ▶ Use of stand-alone LIST policies
- ▶ Use of external scripts

We recommend the use of the integration with external scripts. This option is simpler from an operational perspective because it requires only one step. However, the external script must be implemented, tested, and maintained.

The use of stand-alone LIST policies requires two steps and gives the administrator more control of the file identification and transfer process. The administrator can inspect the file list that is generated by the LIST policy and use customized transfer options in accordance with the file list.

4.4 Use of snapshots

The use of snapshots as source for the data transfer is useful because it prevents file access conflicts between the user and the IBM Aspera Sync processes. In addition, the snapshot can be used to restore files and directories locally.

Unlike with IBM Spectrum Scale Active File Management Disaster (AFM DR) recovery (see 1.3, “Differentiation with IBM Spectrum Scale Active File Management” on page 3), snapshots are not created automatically by IBM Aspera Sync. Therefore, more operational processes are required to coordinate the snapshot creation and deletion and the file transfer.

IBM Aspera Sync does not automatically create snapshots on the target, unlike AFM DR. This process can be done manually or semi-automated in a script. Having matching snapshots on the source and on the target provides defined recovery points.

4.5 Use cases

The integration of IBM Aspera Sync with IBM Spectrum Scale facilitates different use cases, such as file sharing, file migration, and disaster recovery. It is not our objective to explain all the use cases in more detail here; instead, we provide some overall ideas and guidance.

4.5.1 File sharing

IBM Aspera Sync can be used to share specific subset of files between IBM Spectrum Scale clusters that are separated across large distances by efficiently using the WAN bandwidth and preserving file attributes and ACL.

For this use case, `async` can be used standalone by providing file lists. If the file to be shared can be identified by the policy engine, `async` or `ascp` can be integrated with the policy engine.

4.5.2 File migration

IBM Aspera Sync can be used to efficiently copy files from one IBM Spectrum Scale file system to another for migration purposes. The IBM Spectrum Scale file system can also be within the same cluster. This process facilitates the migration of files across different IBM Spectrum Scale file system levels.

The use of `async` standalone assures that the source and the target file system are synchronized in an asynchronous way. In addition, `async` copies in incremental forever fashion and manages deleted and renamed files on the source.

For large file systems on the source, it might be beneficial to integrate `async` with the IBM Spectrum Scale policy engine to accelerate the file identification process. This process also prevents the crawling of `async` through the file system structure, which can cause excessive metadata I/O.

4.5.3 Disaster recovery

IBM Aspera Sync can be used for disaster recovery by synchronizing the content of the source file system with the target file system in an asynchronous manner, especially if the distance between the source and the target file systems is large. The goal of disaster recovery is to have the identical content on the source and the target file system at a certain time.

For the disaster recovery use case, async can be used standalone, which ensures that all files are replicated and deleted and renamed files are managed.

If the duration of the file identification process become too long, it is recommended to integrated async with the IBM Spectrum Scale policy engine. The most efficient way is to implement an external script that is directly started by the policy engine.

To ensure that the content on source and target is identical, it is recommended to perform periodic full synchronizations by running async standalone periodically. This process manages files that were missed for transfer because of network or policy issues. For example, if the policy to transfer files is run every 4 hours, it might be appropriate to run a full synchronization by using async standalone once a day or once a week. During the full synchronization runs, async standalone crawl the file system; therefore, the process can take some time.

One of the key operational aspects for disaster recovery are the failover and failback processes. These processes must be designed, documented, and implemented according to the infrastructure and operational environment.

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this paper.

IBM Redbooks

The IBM Redbooks publication *IBM Spectrum Scale Immutability Introduction, Configuration Guidance, and Use Cases*, REDP-5507, provides more information about the topics that are in this document. Note that this publication might be available in softcopy only.

You can search for, view, download, or this document and other Redbooks, Redpapers, Web Docs, draft, and other materials, at the following website:

ibm.com/redbooks

Online resources

The following websites are also relevant as further information sources:

- ▶ IBM Spectrum Scale overview:

https://www.ibm.com/support/knowledgecenter/en/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b11ins_intro.htm

- ▶ IBM Spectrum Scale Application Programming Interface documentation:

https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b11adm_intrfce.htm

- ▶ Spectrum Scale Active File Management for Disaster Recovery:

https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b11ins_activefilemanagement.htm

- ▶ Configuring IBM Spectrum Protect for Space Management with IBM Spectrum Scale AFM:

<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Tivoli%20Storage%20Manager/page/Configuring%20IBM%20Spectrum%20Scale%20Active%20File%20Management>

- ▶ Reference for the open source tool rsync:

<https://rsync.samba.org>

- ▶ Implementation of rsync for IBM Spectrum Scale:

<https://github.com/gpfsug/gpfsug-tools/tree/master/bin/rsync>

- ▶ IBM Aspera Sync command reference version 3.9 (Linux):

https://download.asperasoft.com/download/docs/entsrv/3.9.1/cs_admin_linux/webhelp/index.html#dita/sync.html

- ▶ Include and Exclude rules with async:
https://download.asperasoft.com/download/docs/entsrv/3.9.1/cs_admin_linux/webhelp/index.html#dita-sync/filtering_rules.html
- ▶ Aspera ascp command reference version 3.9 (Linux):
https://download.asperasoft.com/download/docs/entsrv/3.9.1/cs_admin_linux/webhelp/index.html#dita/ascp_2.html
- ▶ IBM Spectrum Scale configuration parameter for performance tuning and optimization:
https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.2/com.ibm.spectrum.scale.v5r02.doc/b11adm_tuningguide.htm
- ▶ The mmapplypolicy command:
https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b11adm_mmapplypolicy.htm
- ▶ IBM Spectrum Scale policies:
https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b11adv_policies.htm
- ▶ Practical guide for using IBM Spectrum Scale policies:
<http://w3.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102642>
- ▶ IBM Spectrum Scale immutability Redpaper:
<http://www.redbooks.ibm.com/abstracts/redp5507.html>
- ▶ Github: A set of scripts to use the IBM Spectrum Scale policy engine to select files and transfer them using Aspera tools:
<https://github.com/IBMRedbooks/Integration-of-IBM-Aspera-Sync-with-IBM-Spectrum-Scale>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



REDP-5527-00

ISBN 0738457493

Printed in U.S.A.

Get connected

