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

Nils Haustein
Jose M Gomez
Benjamin C Forsyth
Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (February 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 February 26, 2019.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notices</td>
<td>v</td>
</tr>
<tr>
<td>Trademarks</td>
<td>vi</td>
</tr>
<tr>
<td>Preface</td>
<td>vii</td>
</tr>
<tr>
<td>Authors</td>
<td>vii</td>
</tr>
<tr>
<td>Now you can become a published author, too!</td>
<td>viii</td>
</tr>
<tr>
<td>Comments welcome</td>
<td>viii</td>
</tr>
<tr>
<td>Stay connected to IBM Redbooks</td>
<td>viii</td>
</tr>
<tr>
<td>Chapter 1. IBM Aspera sync and IBM Spectrum Scale Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.1 IBM Aspera sync</td>
<td>2</td>
</tr>
<tr>
<td>1.2 IBM Spectrum Scale</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Differentiation with IBM Spectrum Scale Active File Management (AFM)</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Performance measurements</td>
<td>6</td>
</tr>
<tr>
<td>Chapter 2. Setting up IBM Aspera sync with IBM Spectrum Scale</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Preparation</td>
<td>10</td>
</tr>
<tr>
<td>2.2 How async works</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Async example</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 The Sync Database</td>
<td>12</td>
</tr>
<tr>
<td>2.2.3 Sessions</td>
<td>12</td>
</tr>
<tr>
<td>2.2.4 Replication modes</td>
<td>13</td>
</tr>
<tr>
<td>2.2.5 Control overwrites</td>
<td>13</td>
</tr>
<tr>
<td>2.2.6 Control deletion on target</td>
<td>14</td>
</tr>
<tr>
<td>2.2.7 Using file lists</td>
<td>14</td>
</tr>
<tr>
<td>2.2.8 Cluster awareness</td>
<td>15</td>
</tr>
<tr>
<td>2.3 How ascp works</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Ascp example</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2 Transfer modes</td>
<td>18</td>
</tr>
<tr>
<td>2.3.3 Control overwrites</td>
<td>18</td>
</tr>
<tr>
<td>2.3.4 Control deletes on target</td>
<td>19</td>
</tr>
<tr>
<td>2.3.5 Different path names on source and target</td>
<td>19</td>
</tr>
<tr>
<td>2.3.6 Using file lists and file pair lists</td>
<td>20</td>
</tr>
<tr>
<td>2.3.7 Cluster awareness</td>
<td>24</td>
</tr>
<tr>
<td>2.4 IBM Spectrum Scale configuration</td>
<td>24</td>
</tr>
<tr>
<td>Chapter 3. Using IBM Aspera sync with IBM Spectrum Scale</td>
<td>27</td>
</tr>
<tr>
<td>3.1 IBM Spectrum Scale policy engine considerations with IBM Aspera</td>
<td>28</td>
</tr>
<tr>
<td>3.2 Async without policy engine</td>
<td>28</td>
</tr>
<tr>
<td>3.3 Ascp without policy engine</td>
<td>29</td>
</tr>
<tr>
<td>3.4 Integration with the IBM Spectrum Scale policy engine</td>
<td>30</td>
</tr>
<tr>
<td>3.4.1 IBM Spectrum Scale policy engine</td>
<td>31</td>
</tr>
<tr>
<td>3.4.2 Integration with async</td>
<td>37</td>
</tr>
<tr>
<td>3.4.3 Integration with ascp</td>
<td>40</td>
</tr>
<tr>
<td>3.5 Integration with Hierarchical Storage Management (HSM)</td>
<td>44</td>
</tr>
<tr>
<td>3.5.1 Async with HSM on source</td>
<td>45</td>
</tr>
<tr>
<td>3.5.2 Async with HSM on target</td>
<td>48</td>
</tr>
<tr>
<td>3.6 Integration with immutable files</td>
<td>50</td>
</tr>
<tr>
<td>3.7 Integration with IBM Spectrum Scale Snapshots</td>
<td>52</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.7.1 Considerations with ascp and snapshots</td>
<td>54</td>
</tr>
<tr>
<td>3.7.2 Using the policy engine with snapshots</td>
<td>55</td>
</tr>
<tr>
<td>Chapter 4. Summary</td>
<td>57</td>
</tr>
<tr>
<td>4.1 Async or ascp standalone</td>
<td>58</td>
</tr>
<tr>
<td>4.2 Async or ascp integrated with the policy engine</td>
<td>58</td>
</tr>
<tr>
<td>4.3 Standalone LIST policies or external policy script</td>
<td>58</td>
</tr>
<tr>
<td>4.4 Using snapshots</td>
<td>58</td>
</tr>
<tr>
<td>4.5 Use cases</td>
<td>59</td>
</tr>
<tr>
<td>4.5.1 File sharing</td>
<td>59</td>
</tr>
<tr>
<td>4.5.2 File migration</td>
<td>59</td>
</tr>
<tr>
<td>4.5.3 Disaster recovery</td>
<td>59</td>
</tr>
<tr>
<td>Related publications</td>
<td>61</td>
</tr>
<tr>
<td>IBM Redbooks</td>
<td>61</td>
</tr>
<tr>
<td>Online resources</td>
<td>61</td>
</tr>
<tr>
<td>Help from IBM</td>
<td>62</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Redbooks (logo)</th>
<th>IBM®</th>
<th>IBM Spectrum Scale™</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspera®</td>
<td>IBM Spectrum™</td>
<td>Linear Tape File System™</td>
</tr>
<tr>
<td>FASP®</td>
<td>IBM Spectrum Archive™</td>
<td>Redbooks®</td>
</tr>
<tr>
<td>GPFS™</td>
<td>IBM Spectrum Protect™</td>
<td>Redpaper™</td>
</tr>
</tbody>
</table>

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 the majority of data stored in file systems solutions to make this data globally available become more and more important. Files in file systems can be protected or shared by replicating these to another file system located in a remote location. The remote location may be just around the corner or it may be located in a different country or continent. Hence the techniques to protect and share files must take into account long distances, slow and unreliable Wide Area Network connections.

IBM® Spectrum Scale is an extremely scalable clustered file system that can be used to store all kind of unstructured data. It provides open data access via 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™ will explain how IBM Aspera sync can be used to protect and share data stored in IBM Spectrum™ Scale file systems across large distances of several hundred to thousands of miles. We explain the integration of IBM Aspera sync with IBM Spectrum Scale™ and differentiate it from solutions built into IBM Spectrum Scale for protection and sharing. We also elaborate on 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, USA. 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
Integration of IBM Aspera Sync with IBM Spectrum Scale

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:

► Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html
Chapter 1. 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). There are also some in-house performance measurements to show the benefits.
1.1 IBM Aspera sync

For all the amazing technological progress 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 has become more pervasive and daunting across industries with the exponential growth of data generated globally.

IBM Aspera solutions are designed to help you globally ingest, distribute and synchronize huge 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 has invested in the integration with IBM Spectrum Scale to perform file transfer between file systems residing in different IBM Spectrum Scale clusters that are located in different locations. The key benefits of using IBM Aspera with IBM Spectrum Scale are:

- Fast and secure file transfer
- Consistent transfer times for given volumes of data
- 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 providing a comprehensive set of storage services [1]. 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 accessed via Storage Area Networks (SAN). Files 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. A file system can be partitioned into filesets. From a user perspective a fileset is a directory in the file system. From an administrative perspective a fileset can be configured with additional functions such as snapshots, quota, active file management (AFM) and immutability. So instead of creating a snapshot for an entire file system the administrator can create a snapshot for a partition of the file system - a fileset - 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 as well 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 is comprised of a set of nodes that provide access to the underlying storage. This set of nodes can be configured for high availability, tolerating outages of a minority of nodes. While IBM Spectrum Scale provides integrated ways for protecting and sharing files, there are scenarios where these cannot be used. In these cases
IBM Aspera sync can be integrated with IBM Spectrum Scale in order to perform the large distance file transfer at the best achievable speed (see 1.3, “Differentiation with IBM Spectrum Scale Active File Management (AFM)”).

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 and preserves these at the target file system.

**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 manage these attributes [2]. This means when a file is copied from a source to a target directory using standard POSIX command then these extended attributes of the file are not preserved on the target. Hence 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 using the IBM GPFS™ API call `gpfs_fgetattr()` , copies these attributes as a blob to the target system and applies these attributes to the file storage in the target directory using the GPFS API call `gpfs_fputattr()` . Remarkably, IBM Aspera sync also preserves immutability attributes of files stored in an immutable file set.

1.3 Differentiation with IBM Spectrum Scale Active File Management (AFM)

IBM Spectrum Scale provides the Active File Management function (AFM) that can be used to protect and share files globally. One of the AFM flavours is AFM for Disaster Recovery (AFM DR) that can be used to replicate files from a source directory to a target directory [3]. 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) as well as the IBM Spectrum Scale management API.

Both IBM Spectrum Scale AFM DR and IBM Aspera sync are complementary tools to replicate files between 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. There are some conditions where AFM DR cannot be used, but IBM Aspera sync can be implemented. Find below a summary of use cases where IBM Aspera sync can be preferred over AFM DR:

- The source directory includes immutable files and their immutability attributes should be preserved in the target directory
  - IBM Aspera sync preserves immutability attributes of files in the target
  - AFM DR does not yet support immutable filesets
- 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 has limited support for file systems that are managed by HSM [4].

► The distance between the source and the target system is very large and the network connection has limited bandwidth or noticeable latency.
- IBM Aspera sync is optimized for data transfer across long distances across weak network connections
- AFM DR is not optimized for file transfer across long distances over weak networks

► The workload 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 whole files. So even if a small portion of a large file has changed, the next transfer will copy the entire large file.
- AFM DR tracks all changes to files and blocks in the source directory in real time and only transfers blocks of a file that have changed since the last transfer. This helps to save bandwidth but may cause AFM DR to become overloaded if the changes to files and blocks are extremely 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
- While 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 has 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.

Quick summary of the AFM DR characteristics

► AFM DR is an integrated component of IBM Spectrum Scale and is included in the Data Management Edition license.
- AFM DR configuration, operation and monitoring is included in IBM Spectrum Scale
- AFM DR has integrated processes and tools to facilitate failover and failback
- AFM DR transfers changed blocks and not necessarily the entire file again. For example, if for a 1 GB file that has been transferred already a 1 MB block changes, 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 snapshot generated on the source and on the target directory, providing 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 may cause contentions in the source file system.

AFM DR is not supported on fileset where IBM Spectrum Archive, IBM Spectrum Protect for Space Management or IBM Spectrum Scale Transparent Cloud Tiering perform space management.

AFM DR requires fileset as source and target directories. A fileset is partition of the file system that allows additional functions such as AFM DR. From a user perspective a fileset is a directory. From an administrator perspective the fileset can be configured with additional properties.

Quick summary of the IBM Aspera sync characteristics

- 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 filesets where IBM Spectrum Archive or IBM Spectrum Protect for Space Management or IBM Spectrum Scale Transparent Cloud Tiering perform space management. This however requires the integration with the IBM Spectrum Scale policy engine to identify files that are not migrated.
- IBM Aspera sync does not require filesets on the source and the target directory.
- IBM Aspera sync shows the progress of file transfers in the command line interface and via the optional IBM Aspera console.
- IBM Aspera sync does not create snapshots on the source and on the target automatically and does not provide consistent recovery points. However, IBM Aspera sync can use a snapshot as the source for the data transfer; this has to be created manually, however.
- IBM Aspera sync when used in standard mode crawls through the source directory to identify new and modified files 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 crawling through the source directory.
- IBM Aspera sync integrated with IBM Spectrum Scale can only run on one node for a single transfer session because it is not cluster aware. It does not provide high availability because if the node running the session fails the transfer will fail.
- IBM Aspera sync sessions have to 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 prior to the transferred and stored uncompressed in the target directory, unless compression is specified with the IBM Aspera sync command.

Key IBM Aspera sync use cases are (for more information see Chapter 4, “Summary” on page 57):

- For protecting and sharing files over long distances leveraging the guaranteed performance
- 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 leveraging the checksum and incremental synchronization capabilities of IBM Aspera sync
1.4 Performance measurements

For demonstration purposes we have conducted performance measurements transferring 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 (5700 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. Refer to [5] the standard rsync utility and [6] the rsync patch that works with GPFS extended attributes.

The source and target clusters have been configured on single nodes running on virtual machine. No performance optimization in IBM Spectrum Scale has been done. Each cluster had 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 using the internet. This network included multiple firewalls. Figure 1-1 is an overview of the test setup.

![Test setup](image)

Different test cases have been run multiple times using IBM Aspera's async tool and the open source rsync tool. Each test case transferred a number of files with a constant file size per test case. As shown in the table below the first test case transferred a number of files with a size of 256 MB. With async this transfer took 48 seconds in average while it took rsync 248 seconds. Hence async was 5 times faster. In addition the standard deviation of different test runs was only 2 seconds with async and 142 seconds with rsync.

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

Figure 1-2 shows a plot of the performance measurements into a diagram correlating the file size and the transfer duration.
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 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

IBM Aspera sync version 3.9 and higher supports IBM Spectrum Scale extended attributes and ACL. This means IBM Aspera sync transfers NFSv4 ACLs, user defined attributes and immutability attributes for files and directories from the source file system and applies these to the files and directories in the target file system. Both - source and target systems - must be IBM Spectrum Scale clusters 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 2.2, “How async works” on page 10). It keeps track of the files and directories it has replicated in a pair of Sync databases stored in the source and the target file system. This information is used during the async session to determine which files need to be transferred, it also contains checksum information for files that have been transferred. The underlying transfer of files is done by the ascp tool (see 2.3, “How ascp works” on page 16). Ascp transfers files from the source to the target directory 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, it receives the file names to be transferred as input parameter through its management API interface.

**Note:** The async tool is the central part of IBM Aspera sync providing the capability to asynchronously replicate the data between a source and target directory. It has all the intelligence to identify the files that have been created, changed, renamed or deleted and orchestrate the transfer using ascp in the backend.

In chapter 2 we explain how the IBM Aspera tools can be integrated with IBM Spectrum Scale and in particular the integration with the IBM Spectrum Scale policy engine. In Chapter 4, “Summary” on page 57 we provide some guidance when to use async or ascp.
2.1 Preparation

In order to leverage IBM Aspera sync with IBM Spectrum Scale the IBM Aspera sync software packages have to be installed and configured on one or more IBM Spectrum Scale cluster nodes of the source and the target cluster. In addition the SSH-keys of the user used for the file transfer need to 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 executed at the same time then multiple pairs of nodes can participate in this transfer whereby each node pair executes the transfer for one pair of source and target directories.

Note: A single IBM Aspera sync session can only run 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.

2.2 How async works

IBM Aspera sync is invoked with the async command [7]. 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 decide whether a file needs to be transferred completely, partially or not at all (see 2.2.2, “The Sync Database” on page 12).

Once invoked async crawls through the specified source directory (local directory) and identifies all files and directories that have been created, deleted, renamed and modified since the last async run for this session by evaluating the time stamps of the files and matching this to the Sync Database. File are only transferred entirely, async does not perform changed blocks tracking to only transfer changed blocks. For the file transfer async uses ascp in the backend. For files or directories where only the metadata has 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 there are small changes on large files on the source then the entire large file is transferred again. This 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 using IBM Spectrum Scale AFM because this only transfers changed blocks.

If during a transfer session the network connecting the source and target system fails then async may fail as well. However, the next session will pick up where the previous one has failed because the status of the file transfers is tracked in the Sync Database.
2.2.1 Async example

Let's take a look at an async command line example [7] 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 remotename in this example and the data is transferred 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:

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

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

Some important options:

- **-N session-name**  
  Session name (should be the same for a given replication pair)

- **--preserve-xattrs=native**  
  transfers extended attributes (required for IBM Spectrum Scale)

- **--preserve-uid**  
  preserve user ID

- **--preserve-gid**  
  preserve group ID

- **--preserve-access-time**  
  preserve last access time stamp (important for immutable files)

- **--preserve-time**  
  preserve time stamps for mtime

- **--create-dir**  
  create the destination directory if this does not exist

The async command above has 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 fileset or a normal directory and likewise the target directory given with parameter -r. When files are used as the source and should be preserved as filesets on the target then the filesets on the target have to be manually created prior to starting the replication session.

An IBM Spectrum Scale fileset is partition of the file system that allows additional functions such as immutability. From a user perspective a fileset is a directory. From an administrator perspective the fileset can be configured with additional properties. Unlike a normal directory a fileset is created and configured with a set of IBM Spectrum Scale commands. For example to create a fileset named **fsetname** in file system named **fsname** use the command:

```
# mmcrfileset fsname fsetname
```

To link this fileset to a directory in the file system use this command:

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

To make the fileset immutable use this command:

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

**Important:** Async does not preserve filesets on the target. These have to 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 with parameter \(-r\) and includes the user and hostname of the target cluster. The user and host names can also be given with the parameters:

\[-\text{user} \text{ username}\]
\[-\text{host} \text{ hostname}\]

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

### 2.2.2 The Sync Database

Each async session has a name denoted by async command line parameter \(-N\) (see 2.2.1, “Async example” on page 11). For each session name async creates a Sync Database (snap.db) that is stored on both 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 as well as in-progress transfers (a transfer cache for pending files) in the Sync Database.

The default location of the Sync Database in within the source and target directories. According to the example above the database is stored in `/fs1/.private-asp` on the source directory and in `/target/fs1/.private-asp` on the target. Hence 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 using the following parameters:

\[-\text{local-db-dir}=\text{ldbdir}\]
\[-\text{remote-db-dir}=\text{rdbdir}\]

**Note:** As long as the session name for a given 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 controlled by the parameters \(--\text{local-db-dir}\) and \(--\text{remote-db-dir}\) for a given session name and pair of source and target directory name must be the same. In addition, 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. Keep in mind that one async session shall only be run on one IBM Spectrum Scale cluster node. The next transfer session however can be run on a different node, this requires the Sync Database to be available on a shared file system.

### 2.2.3 Sessions

Each async session must have a name that is specified with the parameter \(-N\) of the async command (see 2.2.1, “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 of time. The content of this database is queried and updated during an async session. This means that subsequent async session with the same session
name use the same pair of databases (source and target) and leverage existing file states and checksums.

**Note:** Do not change the name of the session for a given replication relation because changing the session name will create a new Sync database and it will not leverage historical 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 should always be the same. If the session name for a given replication relation is changed then async will recalculate the checksum information for all files in the source and target directory again in order to create a new pair of databases. If files on the source or on the target are migrated by IBM Spectrum Archive EE, IBM Spectrum Protect for Space Management or IBM Spectrum Scale Transparent Cloud Tiering this will cause recalls of all files!

### 2.2.4 Replication modes

Async has three modes of synchronization: push, pull, and bidi. The mode can be specified with the parameter:

```
-K push|pull|bidi
```

With **push** mode the contents of source directory are synchronized to target directory whereby contents of the target directory are overwritten with the contents of the source directory (default behavior, overwrites can also be controlled). Hence 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 default.

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

With **bidi** (bi-directional) mode the contents of source and target directories are synchronized, with newer versions of files and directories overwriting older versions in either the source or target directory (default behavior, overwrites can also be controlled). Since both source and target need to be writable the replication cannot be done from snapshots. When using immutable filesets on source and target conflicts may occur when file names on source and target are the same.

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

### 2.2.5 Control overwrites

The behavior of overwriting files in the target directory can be controlled with overwrite policies:

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

The value of **always** means that a file on the target is overwritten if it has been modified on the source. This value is default.
The value of **older** means that the file on the target is overwritten if it is older than on source. This requires the use of `-t` (**--preserve-time**).

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

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

### 2.2.6 Control deletion on target

By default async deletes files in the target directory that have been 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 have been deleted in the source directory by using the following parameter:

`--ignore-delete`

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

In most cases the deletion of files in the target directory during the replication session is sufficient. There are some rare cases where files on the target should not be deleted (use parameter `--ignore-delete`). However, this causes that the target directory consumes more storage capacity than the corresponding source directory.

### 2.2.7 Using file lists

Async allows transferring files with their names specified in file list. This is useful when integrating IBM Aspera sync 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 for this [8]. When providing include and exclude lists async will crawl through the source directory and identify files matching include and exclude patterns and then check if the files have been modified since the last async run.

In order to include files with their names provided in a file list the use parameter:

`--include-from=filelist`

To include individual file names use parameter:

`--include filename`

Likewise individual file names or file names provided in a file list can be excluded using these parameters:

`--exclude-from=filelist`

`--exclude filename`

File list including file and directory names to be included must be strictly sorted and include separate directory and file names relative to the source directory. For example, in order to create a file list, Example 2-2, for the following files, and directories stored in the source directory named `/fs1`:
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 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, and the first matching rule (whether including or excluding) takes precedence.

Many more include and exclude samples using pattern matching can be found the async man page.

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

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 include and exclude parameters are given in the command above is important for the matching of the file names.

If the order of the parameters --include-from=filelist --exclude "*" would be switched then every file would be excluded.

Transferring files with their names 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).

2.2.8 Cluster awareness

In this section we elaborate on cluster awareness. A cluster is a set of nodes. In typical IBM Spectrum Scale configurations there are multiple nodes providing access to the global file systems. Programs like async could be installed and executed 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 is no.

Async uses the local and remote database to track the status of files and select files for transfer based on the checksums. This means each async process needs to have access to this database. If async runs simultaneously on multiple nodes, replicating data for the same session then these async processes access the same database from different nodes. This has not been tested and is not recommended. It is recommended to run async for a given async session on one node. In order 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, as long as the database is stored in a shared file system. The concurrent access to the Sync DB from different nodes could result in an error being generated by async as it will see the process on the other node has a file lock 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 is not allowed to run on ESS I/O nodes. Async can either be run on dedicated nodes or on protocol nodes providing NFS, SMB, Object, or HDFS access.

### 2.3 How ascp works

Ascp is the backend of async and performs the file transfer from the source to the target leveraging IBM Aspera's leading file transfer techniques [9]. Ascp does not identify files subject for transfer, it just takes the file names as input parameter and transfers these files to the specified target directory. Ascp transfers files entirely.

Ascp can be invoked independent of async. It does not require the Sync Database. When using async after using ascp for the same pair of source and target directories then async does not know about the metadata of the files transferred by ascp. Consequently, async will have to inventory all files copied by ascp and determine their checksum and state in the source and target directory.

**Note:** The combination of using ascp with async and async should be carefully planned because ascp transfers files without providing information about these files to async. There might be use cases where ascp is run periodically like multiple times a day to copy files from the source to the target. Async can be run less frequently (like once a week on the weekend) to assure consistency, just 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 has changed on the file. If only the extended attributes of a file have been changed by the file owner then ascp will apply these attributes only if the file is forced to transfer by specifying the `--overwrite=always` option.

**Note:** If only extended attributes have changed on a file in the source directory and the file is transferred using ascp then 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 connecting the source and target system fails, then ascp may fail as well. Since ascp does not track the status of the file transfer in the Sync Database the next ascp transfer job will copy all files again, that are given in the file list.

### 2.3.1 Ascp example

In the ascp Example 2-5 the contents of the source directory `/fs1` is copied to the target directory `/target/fs1` of the remote cluster [9]. The host name of the remote cluster node is `remotehost` in this example and the data is transferred 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:

*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/
```

Some important options:

- `--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 a IBM Spectrum Scale file system, a fileset or a normal directory and likewise the target directory. When filesets are used as the source and need to be retained on the target then the filesets on the target have to be manually created prior to starting the replication session. The user ID, group ID, time stamps and IBM Spectrum Scale extended attributes are preserved on the target. Ascp also allow to copy single files from the source directory.

An IBM Spectrum Scale fileset is partition of the file system that allows additional functions such as immutability. From a user perspective a fileset is a directory. From an administrator perspective the fileset can be configured with additional properties. Unlike a normal directory a fileset is created and configured with a set of IBM Spectrum Scale commands. For example to create a fileset named `fsname` in file system named `fsname` use the command:

```
# mmcrfileset fsname fsetname
```

To link this fileset to a directory in the file system use this command

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

To make the fileset immutable use this command:

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

**Note:** Ascp does not preserve filesets on the target. These have to be manually created on the target.

In the example above the target directory notation includes the user and host names of the target cluster. The user and host names can also be given with the parameters:
--user username
--host hostname

In the next sections we discuss some more parameters that can be used to control the ascp.

### 2.3.2 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 is default. With recv ascp receives files from the target into the source directory. When using `--mode` the parameter `--host` must be specified. In most cases the mode send is used.

### 2.3.3 Control overwrites

Overwriting files existing on the source and on the target can be controlled with the following parameter:

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

The value of never specifies that existing 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) may still be updated.

The value of always specifies that existing 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 default.

The value of diff+older means that the file on the target is overwritten if it is older AND also different than the source file. For example, if the file on the target is the same as the source, but with a different timestamp, it will not be overwritten. Plus, if the file on the target is different than the source, but newer, it will not be overwritten.

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

The overwrite methods diff or diff+older can be further controlled with the resume policy (parameter `-k`):

- 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 using the overwrite method diff or diff+older assures that only files are transferred that have changed in the source directory. This can help to save network resources. When utilizing the resume policy please keep in mind that the calculation of the checksums takes additional time.
2.3.4 Control deletes on target

By default ascp does not delete files in the target directory that have been deleted in the source directory. In order to delete files in the target that have been deleted on source use the following parameter:

`--delete-before-transfer`

With this parameter ascp identifies and deletes files on the target that do not exist on the source prior to transferring files from the source to the target directory. Of course this only applies to file names 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:

```
# 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.

2.3.5 Different path names on source and target

When using ascp 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 let's copy the source path `/fs1/Documents` to the target path `/target/fs1` using the command shown in Example 2-7:

```
# 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 shown in Table 2-1:

<table>
<thead>
<tr>
<th>On Source</th>
<th>On Target without --src-base=/fs1/Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/fs1/Documents/file1</code></td>
<td><code>/target/fs1/Documents/file1</code></td>
</tr>
<tr>
<td><code>/fs1/Documents/dir1/file2</code></td>
<td><code>/target/fs1/Documents/dir1/file2</code></td>
</tr>
<tr>
<td><code>/fs1/Documents/dir2/file3</code></td>
<td><code>/target/fs1/Documents/dir2/file3</code></td>
</tr>
</tbody>
</table>

It is also possible to strip specified path prefix 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 need to be copied to the target path `/target/fs1`. Thereby, the path name of the files copied to the target should be relative to the source path `/fs1/Documents`. This means the directory Documents should not appear in the target folder. In order to achieve this the following ascp command line option can be used:
Integration of IBM Aspera Sync with IBM Spectrum Scale

--src-base=source-path-prefix

This will strip 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 stored in the source directory /fs1/Documents by eliminating the directory name /Documents in the target directory, set the --src-base=/fs1/Documents with the ascp command as 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 in Table 2-2:

<table>
<thead>
<tr>
<th>On Source</th>
<th>On Target with --src-base=/fs1/Documents</th>
<th>On Target without --src-base=/fs1/Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>/fs1/Documents/file1</td>
<td>/target/fs1/file1</td>
<td>/target/fs1/Documents/file1</td>
</tr>
<tr>
<td>/fs1/Documents/dir1/file2</td>
<td>/target/fs1/dir1/file2</td>
<td>/target/fs1/Documents/dir1/file2</td>
</tr>
</tbody>
</table>

Note: Sources located outside the source base are not transferred. No errors or warnings are issued, but the skipped files are logged. For example, if /fs1/file4 were included in the above example sources, it would not be transferred because it is located outside the specified source base (/fs1/Documents/).

Stripping off path name prefixes can also be used with file list or file pair lists, see “File lists” for more details.

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

2.3.6 Using file lists and file pair lists

Ascp can also transfer files with their names provided in file lists. There are two options to use file lists with ascp: file list including the source path and file names (see , “File lists” on page 20) and file lists that include pairs of source and target path and file names (see “File pair lists” on page 22). In addition to this it is possible to strip off a specified prefix of the source path when files are stored in the target directory (see section 2.3.5, “Different path names on source and target” on page 19).

File lists
To use ordinary file list 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 file **filelist** is a text file including 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.

For example, if **filelist** contains the following list of source files and directories in Example 2-9:

```
Example 2-9  List of source files and directories
/tmp/code/compute.php
doc_dir/
images/iris.png
images/rose.png
```

And the following command 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 /
```

Then the target, in this case the transfer user's docroot (configuration option specifying the area of the file system the user has access to), will contain the following shown in Example 2-11:

```
Example 2-11  Command results
compute.php
doc_dir/ (and its contents)
images/iris.png
images/rose.png
```

All directory prefixes of the path and file names on the source are removed on the target. In order to keep path names on the target use the parameter `--src-base`. For example to keep the directory structure of the files in the file list on the target use the following command 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 will be identical to the source shown in Example 2-13:

```
Example 2-13  Target directory structure will be identical to the source
/tmp/code/compute.php
doc_dir/
images/iris.png
images/rose.png
```
If the source and target directory and file names are identical then the parameter `--src-base` and the target directory should be set to “/” while the file list includes the fully qualified path names shown in Example 2-14.

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

```bash
# 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 using file lists with ascp:

- The command line cannot use the user@host:source syntax. Instead, specify this information with the options `--mode`, `--host`, and `--user`.
- Paths specified in the file list cannot use the user@host:source syntax.
- Because multiple sources are being 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 specified in the file list are transferred; any sources 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 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 including 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. 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 will still be relative to the docroot. Target paths specified in the list are created automatically if they do not already exist.

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

```plaintext
Note: the files and directories in the filelist must be located within the `--src-base` value of `/fs1/Documents` on the source. Files not matching the `--src-base` path prefix are not copied.
```
Note: The the line numbers shown in Example 2-15 are given to denote odd and even lines and have to be omitted for the actual transfer.

Example 2-15  filepairlist directory and file names

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

And the following command 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 /
```

Then the destination, in this case the transfer user's docroot, now contains the following 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 using file pair lists:

- The command line cannot use the user@host:source syntax. Instead, specify this information with the options --mode, --host, and --user.
- The user@host:source syntax cannot be used with paths specified in the file list.
- Because multiple sources are being transferred, the destination 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 additional source files specified on the command line are ignored.
- If the source paths are URLs, the file list cannot exceed 24 KB.

Using file lists describing the file names to be transferred is useful when integrating the ascp command with the IBM Spectrum Scale policy engine (see 3.4, “Integration with the IBM Spectrum Scale policy engine” on page 30.
2.3.7 Cluster awareness

An IBM Spectrum Scale cluster is comprised of multiple cluster node. 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 have the limitation outlined for async (see 2.2.8, “Cluster awareness” on page 15). This is because ascp does not use the Sync Database and hence does not have the need to access a database simultaneously from multiple nodes.

When running ascp simultaneously on multiple nodes it has to be assured that the ascp instances do not transfer the same files, because this 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 with the IBM Spectrum Scale policy engine (see 3.4.3, “Integration with ascp” on page 40).

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

2.4 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 should be designed to support the required throughput. In addition, other jobs (user I/O, backup, tiering, etc) that run at the same time as the replication must be taken into account for the disk sizing because these may cause additional I/O.

When using async without the IBM Spectrum Scale policy engine it will cause high metadata I/O on the source file system because it gathers the file stat information. Consider using fast disk storage (SSD or Flash) for the file system metadata.

The following of the IBM Spectrum Scale configuration parameter should be adjusted [10]:

- **maxFilesToCache**: Controls the number of files cached at a time. Since IBM Aspera sync opens many files on the source for transfer the value of this parameter should be large enough and aligned with the async option --pending-max=N that defines the maximum number of files that are pending. This async option acts as a buffer to ensure that files to be transferred do not exceed the maximum; default=2000.

- **maxStatCache**: Controls the number of file attributes to cache in addition to what is cached already in pagepool. When possible set this parameter to the number of files stored in the source directory. If this is not possible 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 running IBM Spectrum Scale version 5.0.1 or below.

- **Pagepool**: Amount of memory used to cache file data and metadata. Since the copy process is sequential both the source and target system can benefit from large pagepools.

- **WorkerThreads**: Controls the number of read and write threads. Especially when using multiple async or ascp threads or processes at the same time the number of worker threads should be increased. The transfer threads with async can be controlled with option --transfer-threads.

To replicate the fileset structure used within the source directory to the target, the filesets have to be created manually in the target file system. IBM Aspera sync has no awareness for
filesets, it considers these directories. If a directory on the target is a manually created fileset then IBM Aspera sync will copy the according files into this fileset.
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. In the Summary chapter we provide some guidance about using these different options.
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 (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, e.g. files that have been created or modified within a certain time period. The policy engine can be programmed with rules expressing the criteria for file identification. The file (and directory) names identified according to the criteria are provided in file lists which can be further processed by IBM Aspera sync. Hence, 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 (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 should be used when a large number of files are stored in the source directory. With a large number of files in the source directory the identification of files subject for transfer may take a long time when using the standard async process because async crawls through the source directory to identify file 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 may take long times. This may cause negative side effects in the IBM Spectrum Scale cluster and the async session may take a long time causing subsequent async sessions to overlap with previous sessions. This can be prevented by integrating async with the IBM Spectrum Scale policy engine (see 3.4.2, “Integration with async” on page 37).

Ascp does not identify files, it gets file and directory names and patterns as input parameters. The file and directory names to be transferred by ascp could 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. Hence integrating ascp with the IBM Spectrum Scale policy engine makes a lot of sense, because the policy engine can efficiently identify new and modified files and pass these file names in lists to the ascp program (3.4.3, “Integration with ascp” on page 40).

In addition IBM Aspera sync can use snapshots as the source for the transfer (see 3.7, “Integration with IBM Spectrum Scale Snapshots” on page 52). There is no native integration with hierarchical storage management tools (HSM) in IBM Spectrum Scale, however if IBM Aspera sync is integrated with the IBM Spectrum Scale policy engine then files that are migrated can be excluded from the file transfer (see 3.5, “Integration with Hierarchical Storage Management (HSM)” on page 44). IBM Aspera sync can transfer immutability attributes of files which is detailed in 3.6, “Integration with immutable files” on page 50).

3.2 Async without policy engine

Async has the capability to identify new, modify and deleted files on the source and transfer these 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.

In order 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. In 2.2, “How async works” on page 10 we have provided a number of examples addressing different requirements. Example 3-1 is an async summary example:
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 using async for a given pair of source and target directories, it is important to always use the same session name (parameter \(-N\) ), because this leverages the existing Sync database (see 2.2.2, “The Sync Database” on page 12). When using a different session name than async will create a new Sync database which can take a tremendous amount of time depending on the number of files in the source and target directory. 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.

The example above uses the push mode (default, see 2.2.4, “Replication modes” on page 13), where files are pushed from source to the target. The overwrite policy is set to always (default, see 2.2.5, “Control overwrites” on page 13) where files that have been identified by the async process to be transferred will overwrite the file instance on the target. Files that have been deleted on the source are also deleted on the target during at the end of the async session, controlled by parameter --delete-delay. By default files 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 have been created or modified in the traditional way. This means it stats each file and directory in the source directory to identify files that have been 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 impact 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 does not have the capabilities to identify files in the source directory that have to be transferred, this is normally done by async. Ascp is the backend of async responsible to transfer files from the source to the target directory. The file and directory names to be transferred are provided by async.

Async has all the optimizations to identify files that need to be transferred and invokes 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 (see 3.4.3, “Integration with ascp” on page 40).

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 need to be transferred in a non-automated environment. When doing this, please keep in mind that ascp does not update the Sync database that is managed by async (see 2.2.2, “The Sync Database” on page 12). Thus certain information about files that have been transferred bypasses the async controls. Example 3-2 is a summary example for using ascp (see 2.3, “How ascp works” on page 16):
Example 3-2  ascp command example

```bash
# 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/ [--overwite=diff -k 1 --delete-before-transfer]
```

The example above simply copies files from the source directory `/fs1` to the target directory `/target/fs1` and preserves the IBM Spectrum Scale extended attributes as well as the user-ID and group-ID. Files on the target are only transferred and overwritten if they have a different file size than on source (parameters `--overwite=diff -k 1`, see 2.2.5, “Control overwrites” on page 13). The parameter `--delete-before-transfer` takes care to delete files in the target directory that had been deleted in the source directory (see 2.3.4, “Control deletes on target” on page 19).

Keep in mind that ascp does not update the Sync Database with the file it has copied. Consequently a subsequent async run for the same files and directories will have to inventory all files copied by ascp and determine their checksum and state in the source and target directory. It might be useful to run ascp periodically like multiple times a day to copy files from the source to the target. Async can be run less frequently (like once a week on the weekend) to assure consistency, just in case ascp was not successful 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, e.g. files that have been created or modified within a certain time period. The policy engine can be programmed with rules expressing the criteria for file identification (see 3.4.1, “IBM Spectrum Scale policy engine” on page 31). The file (and directory) names identified according to the criteria are provided in file lists which can be further processed by IBM Aspera sync. Hence, 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.

Both IBM Aspera sync tool - async (see 3.4.2, “Integration with async” on page 37) and ascp (see 3.4.3, “Integration with ascp” on page 40) - can be integrated with the IBM Spectrum Scale policy engine. The difference between both is that async keeps track of the status of files and directories in the source and target directory that it transfers, ascp does not. Of course this comes for a price: async has more overhead. However, async does not crawl through the file system to identify files subject for transfer but instead picks the file names provided by the policy engine in one or more file lists.

There are two ways to integrate IBM Aspera sync with the policy engine: Using standalone LIST policy and using a LIST policy with an external script. In common for both is that the policy rules for file identification are identical.

When using standalone LIST policy two steps are required. In the first step the IBM Spectrum Scale policy engine is invoked 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 invoked 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 generated by the policy engine. He can also invoke the IBM Aspera sync sessions with different parameters. The disadvantage of this method is that it requires two steps that need to be orchestrated.
When using LIST policy with an external script only one step is required whereby the policy engine is invoked with a LIST policy to create lists of files names according to the criteria and invokes an external script. The external script obtains the list of files identified by the policy engine as input parameter and invokes IBM Aspera sync. The advantage of this method is that it requires just one step. The disadvantage is that the administrator has less control over the file lists and the parameters used for the IBM Aspera sync sessions.

Before we further explain the integration of IBM Aspera sync with the IBM Spectrum Scale policy engine using the two methods explained above we provide a quick overview about the IBM Spectrum Scale policy engine.

### 3.4.1 IBM Spectrum Scale policy engine

The IBM Spectrum Scale policy engine allow fast identification of files based on their attributes. The policy engine is represented by the command: `mmapplypolicy` [11]. This command takes a policy file as input and identifies the files based on the rules included in these policies. The policy file is a plain text file written in the IBM Spectrum Scale policy syntax [12]. The rules define the selection criteria for the files based on file attribute and actions to be performed with the selected files. The actions performed by the policy engine in the context of the integration with IBM Aspera sync are either to provide a file list including the names of the selected files or to invoke 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. We elaborate on the most important parameters:

```
# mmapplypolicy /fs1 -P policyfile -N asperanode -m 1 -B 1000 --single-instance -s localdir -g globaldir -l test|defer|yes [-f fileprefix]
```

The parameters of the commands are the following:

- `/fs1` is the source directory where files need to be identified
- `-P policyfile` is the file including 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 already running then this command will abort
- `-m 1` means that there is just one thread started to process the selected files. In many cases using 1 thread is sufficient on a single node.
- `-B 1000` specifies the number of file names in one file list. In this case it would be 1000 files per file list.
- `-s localdir` specifies a local directory used to store temporary files created by the policy engine. There must be sufficient space in this directory. The default directory is `/tmp`.
- `-g globaldir` specifies a directory that is accessible be all cluster nodes. It can be in the file system that is processed by the policy engine or it can be in a different file system. The default is specified by the IBM Spectrum Scale configuration parameter `sharedTmpDir`.
- `-l test|defer|yes` specifies the mode for the policy run. Test means that the policy is tested for syntax. Defer means that the policy perform file selection
but does not invoke the action. Yes means that the policy performs file selection and executes the action.

- \texttt{f fileprefix} specifies a prefix for an output file that is generated with a list policy when it is run in deferred mode.

The file named \texttt{policyfile} is the policy that includes the rules for the file selection. The following rule, Example 3-4, identifies and selects files that have been created or modified (including file metadata modifications only) since the given time stamp of 2018-06-06 12:37:00.

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

\begin{verbatim}
RULE 'asperaRule' LIST 'files' WHERE
  ( MODIFICATION_TIME >= TIMESTAMP("2018-06-06 12:37:00") OR
    CHANGE_TIME >= TIMESTAMP("2018-06-06 12:37:00") )
\end{verbatim}

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 shown above includes two records with one per line. Each record has 5 columns separated with blanks. The first three columns are IBM Spectrum Scale internal numbers describing the file (\texttt{inodenumber}, \texttt{inodegeneneration}, \texttt{snapid}). The file name is the 5th field in the file list. Each record starts in a new line.

Depending on the policy rules, the policy engine will either create the file lists (see , “Standalone LIST policies” on page 32) or it will invoke an external script and pass the file name of the file lists to this program (see , “LIST policies with external script” on page 33).

With this basic knowledge about IBM Spectrum Scale policies we will explain the concept of standalone LIST policies and LIST policies with an external script in the next sections. Afterwards we explain the integration of IBM Aspera sync with these two methods. For more information and guidance for using the policy engine read this whitepaper [12].

\section*{Standalone LIST policies}

With standalone LIST policies the IBM Spectrum Scale policy engine is used to create a file list including file names that have been identified according to the criteria of the policy and store 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 specifying directory and file names that should be excluded from the identification process. The second rule defines that the policy engine should create a list of files. The third rule specifies the selection criteria of the files that have been created or modified after a certain time stamp (in this example \texttt{TIMESTAMP("2018-06-06 12:37:00")}):
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 have been create or modified (including file
attribute modification) are selected with this rule.

To run this policy use the following mmapplypolicy command in deferred mode shown in
Example 3-6 (see 3.4.1, “IBM Spectrum Scale policy engine” on page 31 for the parameter
used with this command):

Example 3-6   mmapplypolicy command in deferred mode

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

For more explanation about the parameters used with the mmapplypolicy command refer to
3.4.1, “IBM Spectrum Scale policy engine” on page 31). Parameters -B and -m are not
required with standalone LIST policies.

The mmapplypolicy command shown above will create a file list named policy.list.modfiles
whereby the first part of the file name comes from the -f option, the second part is fix and the
third part of this name comes from the EXTERNAL LIST name of rule 2 above. This file list
includes files that have been identified according to the criteria specified in rule 3. The format
of this file list is the following:

48900 1741777473 0   -- /fs1/file1

The three first numbers are GPFS internal numbers (inodenum, inodegeneration, snapid). The file name is the 5th field in the list file. There can be multiple lines whereby each
line contains one record in the format shown above.

This file list can subsequently be adjusted and split into multiple file lists when required and
fed into the IBM Aspera sync tools (see 3.4.2, “Integration with async” on page 37 and 3.4.3,
“Integration with ascp” on page 40).

LIST policies with external script
A LIST policy can also be programmed in a way that it directly invokes an external script that
processes the file lists 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. Example 3-7 shows
a policy with three rules. The first and the third rule are identical to the standalone LIST policy
(see ”Standalone LIST policies” on page 32). The second rule includes the name of the
external script (/usr/local/bin/myscript.sh) that has to be programmed to process the file lists.
Integration of IBM Aspera Sync with IBM Spectrum Scale

The third rule specifies the selection criteria for files that have been created or modified after a certain 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 have been create or modified (including file attribute modification) are selected with this rule.

To run this policy use the command shown in Example 3-8 (see 3.4.1, “IBM Spectrum Scale policy engine” on page 31 for the parameter used with this command):

Example 3-8  Run the mmapplypolicy command

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

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 (specified with parameter -B 1000 in the mmapplypolicy command) and invokes the external script (specified in rule 2, /usr/local/bin/myscript.sh). The policy engine launches one instance of the external script (specified with parameter -m 1 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 then another instance is launched with another file list. The external script is launched on the node 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. It depends 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 being launched in parallel by the policy engine (parameter -m 1 in the mmapplypolicy command) can also be adjusted. When using async in the external script the recommended value for this parameter is 1 because async itself can
launch multiple transfer threads. When using ascp then the value of this parameter can be greater than 1.

The node name specified by parameter -N of the mmapplypolicy command must be a name of node (IBM Spectrum Scale node name) that has the IBM Aspera sync tools installed and configured and a network connection to the target cluster. When using async it is recommended to specify one node name for each async session name. I.e. it is not recommended to run async for a given session name in parallel on multiple nodes. When using ascp multiple node names can be specified in order 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 has to process the file list. Before we explain how to process these file lists with IBM Aspera sync we briefly explain how the external script works in general.

The external script will receive 2 or more parameters from the policy engine:

1. Parameter: string describing the operation, this will be TEST and LIST in this example. The external script is first invoked with 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, e.g. assure IBM Aspera sync is installed.

2. Parameter: depends on 1. Parameter. If the first parameter is LIST then the second parameter is the name of the file list including the file names identified. If the first parameter is TEST then the second parameter is the name of the file system.

3. Optional parameter: given with the OPTS clause in the second rule. In the example above the second rule does not contain an OPTS clause, so in the example below we do not use this set of parameters.

Once the external script is invoked 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 have been identified by the policy engine. The format of the file list is shown below:

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

The three first numbers are IBM Spectrum Scale internal numbers (inodenumber, inodegeneration, snapid). The file name is the 5th field in the file list.

With this file list the external script can extract and process the file names. Example 3-9 shows some pseudo code in korn shell script semantic for the implementation of the external script. Note, based on the string describing the operation in the 1st parameter the 2nd parameter might either 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

# pseudo code 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
```
Integration of IBM Aspera Sync with IBM Spectrum Scale

```bash
#
# 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 non-zero 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
    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
    fi

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

  * )
    echo "Unknown argument $1 received"
    rc=1;;
esac
```
The function process shown in the above pseudo code example needs to be implemented according to the needs. We will show how to implement this function for async (see, “Async with external script” on page 38) and ascp (see, “Ascp with external script” on page 42) separately in the next sections.

### 3.4.2 Integration with async

Async can transfer files from a file list including the file names subject for transfer from source to target as shown in 2.2.7, “Using file lists” on page 14. Async can either be invoked manually with a file list generated by the policy engine with the standalone LIST policy (see, “Async with standalone LIST policies” on page 37) or automatically by the policy engine with an external script (see, “Async with external script” on page 38).

**Important:** When using async with file lists provided by the policy engine then files that have been deleted in the source directory are not deleted in the target directory. Furthermore, files that have been renamed in the source directory are transferred as new files to the target directory and the old file persists in the target directory. Additional measures have to be taken to promote deletions and renames to the target as explained below.

**Async with standalone LIST policies**

To use async in association with standalone LIST policies the policy engine has to be invoked with 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 has been described in, “Standalone LIST policies” on page 32. As a result of this the policy engine produced a file list (policy.list.modfiles) including the path and file names of files that matched the criteria of the policy.

This file list provided by the policy engine has to be adjusted before it can be passed to async with the parameter `--include-from=filelist`. The adjustment of the file list is necessary because the format of the file list created by the policy engine (see, “Standalone LIST policies” on page 32) does not match the format expected by async (see 2.2.7, “Using file lists” on page 14). Async requires the path and file name relative to the source directory while the policy engine provides the complete path and file name relative to the file system root as the fifth column. Hence, the file and path name needs to be extracted from the file list provided by the policy engine and adjusted to be relative to the source directory.

In order to adjust the file list created by the policy engine to match the format expected by async the following command shown in Example 3-10 can be used. The source directory is `/fs1`, the file list provided by the policy engine is named policy.list.modfiles and the file including 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
```
Once the file list produced by the policy has been adjusted to match the async format (file name async.transfer.list in the example above) it can be passed to async using parameter

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

Example 3-11 shows the async command using the adjusted file list:

```
Example 3-11   async command using 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
```

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

In order to promote deleted and renamed files to the target async has to be run without the policy engine. This can be done periodically (e.g. once a week) in times where the workload in the source directory is low. To run async without the policy engine the following command 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 used for the transfer of files from a file list in order to leverage the existing Sync database.

### Async with external script

Async can also be invoked through the policy engine using a LIST policy with an external script (see, “LIST policies with external script” on page 33). The name of the file list 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 invokes async. The invocation of async can be implemented in function process of the example above.

Before async can be invoked within the external scripts function process the file list provided by the policy engine needs to be adjusted (see, “Async with standalone LIST policies” on page 37). The adjusted file list is passed to async via the parameter `--include-from`.

Example 3-13 shows pseudo code example for the function process that can be integrated within the external script /usr/local/bin/myscript.sh explained in, “LIST policies with external script” on page 33:

---

**Note:** The adjustment of the file list provided by the policy engine shown above tolerates files with blanks in their name. However, it does not tolerate file names with multiple consecutive blanks in their name.

**Note:** Consider to split the file list if the number of file names contained in this list is too high to be processed by a single async process. This requires to launch multiple async processes sequentially. Alternatively consider to increase the number of threads launched by async in accordance to the number of files.
Example 3-13  pseudo code 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 shown above can be with the sample external script
/usr/local/bin/myscript.sh shown in , “LIST policies with external script” on page 33.

Note, the extraction of the file names 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.

In order to run async invoked by an external script a policy needs to be provided (find an
example in, “LIST policies with external script” on page 33). This policy is executed with the
mmapplypolicy command as shown in Example 3-14:

Example 3-14  mmapplypolicy command executing policy

```
# mmapplypolicy /fs1 -P policyfile -N asperanode -m 1 -B 1000 --single-instance
[-s localdir -g globaldir]
```
Note that the parameter \(-N\) should specify only one node and the parameter \(-m\) should be set to 1.

This \texttt{mmapplypolicy} command invokes the IBM Spectrum Scale policy engine. The policy engine identifies file based on their modification time, stores the file names in file lists and invoke the external script with one file list at a time. The external script that implemented the function \texttt{process()} shown above will adjust the file list and provide it to async. Async performs further checks on the files provided in the list and transfer the files when required. If one instance of the external script processing one file list has finished another instance of the external script is started with a new file list, until all file lists are processed.

With this implementation files are identified by the IBM Spectrum Scale policy and not by async. If there are many files stored in the source directory the policy engine will be faster than async. Async will process these files identified by the policy engine by determining the file attributes, calculating the checksum and comparing this with the file stored on the target before transferring the file. It will create and maintain a Sync database on the source and the target for this and keep record for each file name it has processed. Therefore it is important to keep the name of the session identical for an existing replication relation.

In order to promote deleted and renamed files to the target async has to be run without the policy engine. This can be done periodically (e.g. once a week) in times where the workload in the source directory is low. To run async without the policy engine the following command shown in Example 3-15 can be used:

\begin{verbatim}
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
\end{verbatim}

The session name (\texttt{sync_fs1} in this example) must match the session name used in the process function (\texttt{\$sessionName}) in order to leverage the existing Sync database.

### 3.4.3 Integration with ascp

Alternatively the files identified by the IBM Spectrum Scale policy engine can be transferred with ascp. As explained in 2.3, “How ascp works” on page 16, ascp is the backend of async and simply transfers the file from the source directory to the target directory under consideration of 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 2.3.6, “Using file lists and file pair lists” on page 20. The tool ascp can either be invoked manually with a file list generated by the policy engine with the standalone LIST policy (see, “Ascp with standalone LIST policies” on page 41) or automatically by the policy engine with an external script (see, “Ascp with external script” on page 42).

\textbf{Important:} When using ascp with file lists provided by the policy engine then files that have been deleted in the source directory are not deleted in the target directory. Furthermore, files that have been renamed in the source directory are transferred as new files to the target directory and the old file persists in the target directory. Additional measures have to be undertaken to promote deletions and renames for files on the source to the target, some measures are explained below.
Ascp with standalone LIST policies

To use ascp in association with standalone LIST policies the policy engine has to be invoked with 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 has been described in “Standalone LIST policies” on page 32. As a result of this the policy engine produced a file list (policy.list.modfiles) including the path and file names of files that matched the criteria.

This file list has to be adjusted before it can be passed to ascp with the parameter --file-list=filelist. The adjustment of the file list is necessary because the format of the file list created by the policy engine (see 3.4.1, “IBM Spectrum Scale policy engine” on page 31) does not match the format expected by ascp (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 provided by the policy engine. Hence, the file and path name needs to be extracted from the file list provided by the policy engine. In order to adjust the file list created by the policy engine to match the format expected by ascp the following command shown Example 3-16 can be used. This essentially extracts the full path and file names from the file list provided by the policy engine (policy.list.modfiles) and stores the resulting path and file names in file ascp.transfer.list:

```bash
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 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 additional fields, for example, when the SHOW statement is used within the policy rules.

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

```bash
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 to split the file list if the number of file names contained in this list is too high to be processed by a single ascp process. This requires to launch multiple ascp processes sequentially.

Ascp will copy each file provided in the file list (ascp.transfer.list) from the source directory (/fs1) to the target directory (/target/fs1) considering the overwrite policies. It will not create and maintain a Sync database.

In order to promote deleted and renamed files on the source to the target ascp has to be for the entire source directory without providing a file list using the policy engine. In addition the ascp parameter --delete-before-transfer has to be used (see 2.3.4, “Control deletes on target” on page 19). You can run ascp this way periodically (e.g. 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 following command shown in Example 3-18 can be used:

```bash
Note:
```
Example 3-18  Run ascp for managing deleted and renamed files

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

This will delete the deleted and renamed files on the target. Other files are copied from the source directory (/fs1) to the target directory (/target/fs1). Overwrite policies can be used to control the processing in the target directory (see 2.3.3, “Control overwrites” on page 18).

Ascp with external script

Ascp can also be invoked through the policy engine using a LIST policy with an external script (see , “LIST policies with external script” on page 33). The name of the file list 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 invokes ascp with an adjusted file list. The invocation of ascp can be implemented in function process of the example above.

Before ascp can be invoked the file list needs to be adjusted because the format of the file list provided by the policy engine does not match the format expected by ascp. This format adjustment has been explained in , “Ascp with standalone LIST policies” on page 41. The adjusted file list is provided to ascp with the parameter --file-list=filelist (see 2.3.6, “Using file lists and file pair lists” on page 20). The pseudo code example in Example 3-19 shows how to invoke ascp with a file list produced by the policy engine.

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

```bash
#!/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"
sessionId="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
```
The function process shown above can be with the sample external script /usr/local/bin/myscript.sh shown in "LIST policies with external script" on page 33.

In order to run ascp invoked by an external script a policy needs to be provided (find an example in, "LIST policies with external script" on page 33). This policy is executed with the `mmapplypolicy` command as shown in Example 3-20:

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

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

The parameter `-m` can have a value of 1 or larger. With this parameter set to 2 as shown in the example above two instances of the external script are started on each node specified with the `-N` parameter in parallel. Each instance of the external script processes one file list.

This `mmapplypolicy` command invokes the IBM Spectrum Scale policy engine. The policy engine identifies file based on their modification time, store the file names in file lists and invoke the external script with one file list at a time. The external script that implemented the function process shown above will adjust the file list and provide it to ascp. Ascp transfers the files named in the file list. If one instance of the external script processing one file list has finished another instance of the external script is started with a new file list, until all file lists are processed.

The integration of ascp with the IBM Spectrum Scale policy engine through an external script is an alternative way instead of using async. With this integration async is not required because the policy engine takes care of identifying the files to be transferred. Using 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 being transferred.

In order to promote deleted and renamed files to the target ascp has to be run without the policy engine and with the parameter `--delete-before-transfer` (see 2.3.4, “Control deletes on target” on page 19). This can be done periodically (e.g. once a week) in times where the workload in the source directory is low. To run ascp without the policy engine in order to manage deleted and renamed files the following command shown in Example 3-21 can be used:

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

**Note:** The extractions of the file names from the file list 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 there are further fields in the file list then this extraction has to be adjusted.
Example 3-21  Run ascp without the policy engine in order to manage deleted and renamed files

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

This will delete the deleted and renamed files on the target. Other files are copied from the source directory (`/fs1`) to the target directory (`/target/fs1`). Overwrite policies can be used to control the processing in the target directory (see 2.3.3, “Control overwrites” on page 18).

In summary, using ascp with the policy engine is simpler and more scalable than async because it can run across multiple nodes. However, renamed and deleted files have to be handled as shown above. In addition it is recommended to periodically check the inventory of the source and the target, just to assure that no files have been missed during replication. For the inventory check of the source and the target file system the IBM Spectrum Scale policy engine could be used to generate lists of files.

### 3.5 Integration with Hierarchical Storage Management (HSM)

IBM Spectrum Scale provides the capabilities to migrate files 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 additional software components are required. This can either be IBM Spectrum Archive Enterprise Edition that migrates files from disk to tape formatted in the IBM Linear Tape File System™ format (LTFS). Or it can be IBM Spectrum Protect for Space Management that migrates files from disk to a IBM Spectrum Protect server which can store the data on tape. This migration capabilities from disk to tape is also called hierarchical storage management (HSM) and provides cost savings by moving data that needs to be retained and is no longer accessed to a cheaper storage tier like 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 have to be migrated IBM Spectrum Scale MIGRATE policies can be used in conjunction with the policy engine [12]. During the migration process the HSM component copies the file to the destination tape via IBM Spectrum Archive or IBM Spectrum Protect and then creates a stub of the migrated file. The stub is essentially the inode of the file which 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 then the HSM component is invoked by IBM Spectrum Scale, identifies the destination of the file and copies the file back to disk. This process is called recall. Once the file content has been completely recalled access is granted to the process that opened the file. The recall from tape takes some time because the tape has to be mounted and spooled. Therefore, it is not recommended to recall many files at one time.

When async identifies files to be transferred it will first check if the file has been modified based on the file modification time stamp. If this is the case then async opens and reads the file in order 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 happens with many files during an async session then many recalls are executed at the same time which can impact the file system performance and the transfer performance of the async session. Many transparent recalls at a time are also called recall storms and can cause hangs in the file system and excessive long async sessions. It may even result in a condition where a subsequent async session kicks in while the previous one is not finished. Recall storms have to be prevented because they are not good.
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 - could be installed and configured on the source and on the target directory. In the subsequent section we explain both configurations and provide some best practices for using async. While we focus on async we will also provide some guidance for ascp.

3.5.1 Async with HSM on source

In this section we primarily focus on async with HSM on the source and provide some guidance for ascp in , "Considerations for ascp" on page 47.

When using async to transfer files from a source directory that is managed by HSM, it must be ensured that the files being transferred have not been migrated to tape prior to this. Otherwise the async will open and read migrated files on the source directory in order to determine the checksum which results in a recall. If this happens to multiple files at the same time then recall storms are inevitable. For this reason, it has to be assured 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 prior to the replication using async, files on the source directory could be migrated with a time delay of some days after their creation or modification. During this time delay async can assure that the file is replicated. Example 3-22 is a migration policy that migrates files 10 days after modification [13]:

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-asf/%' 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
```
An even better way to prevent recall storms caused by async is to integrate async with the policy engine (see 3.4.2, “Integration with async” on page 37). This allows to exclude files that are migrated from being identified as candidates for the async session. Hence, migrated files will not become candidates for replication by async. An example of such 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 the policy above 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 described in 3.4.2, “Integration with async” on page 37.

The condition that async skips migrated files that need to be transferred needs to be resolved in order to have a consistent state between source and target. For this purpose a standalone 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. More information about standalone LIST policies can be found in “Standalone LIST policies” on page 32. Example 3-24 shows a policy that selects files that have been migrated after a specified modification time:

**Example 3-24  Policy that selects files that have been 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 '/.ltfsee/%'
   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 executed with the `mmapplypolicy` command to create a list of files matching the criteria as shown in Example 3-25:

```
Example 3-25  Use 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 file list including the file names of migrated files that have not been transferred yet is named `migrated.list.modfiles`. This file lists can be used to execute bulk recalls 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. Hence the number of tape mounts is drastically reduced and the transfer speed is increased because all files stored on a given tape are recalled in one operation. Find below and example to initiate a bulk recall with IBM Spectrum Archive:

```
# ltfsee recall migrated.list.modfiles
```

When using IBM Spectrum Protect for Space Management the file list created by the policy engine has to be adjusted in order to extract the path and file names. The following command can be used as shown in Example 3-26:

```
Example 3-26  Adjust policy engine in order 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
```

Subsequently the recall can be executed with IBM Spectrum Protect for Space Management. Specifying the file system path of the source directory will perform a tape optimized recall:

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

Once these files are recalled, async can be executed again through the policy engine to transfer these files to the target directory. Once this has finished the files that have been recalled can be migrated again using the migration command of IBM Spectrum Archive (`ltfsee migrate -s filelist`) or IBM Spectrum Protect for Space Management (`dsmmigrate -filelist=filelist`).

In summary when managing the source directory with HSM it is important to assure that files are replicated by async before they are migrated. This can be achieved by delaying the migration of files based on the modification time stamp. In addition it shall be assured that files once stored in the source directory are not changed again in order to prevent recalls and subsequent replication and migration. This can be implemented by setting files to immutable on the source right after the files have been stored. Immutable files cannot be changed. When integrating async with the IBM Spectrum Scale policy engine it can be assured that files that are migrated are not selected to be transferred by async.

**Considerations for ascp**

Ascp does not identify files that have been created and modified in the source directory. Rather than ascp is given a file name or a file list including 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 will be recalled. If this happens to many files at the same time then recall storms are inevitable.
In order to prevent recall storms file names passed to ascp should not be migrated on source. The best way to achieve this is to integrate ascp with the IBM Spectrum Scale policy engine (3.4.3, “Integration with ascp” on page 40). When ascp uses the IBM Spectrum Scale policy engine then the policy rules used to identify files subject for transfer can exclude files that are migrated on the source directory. A policy to exclude files that are migrated on source is shown in 3.5.1, “Async with HSM on source” on page 45.

Another way to prevent recall storms using ascp is to prevent overwrites using the parameter:

\[ --overwrite=never \]

However this will cause inconsistencies between source and target directories because file that are modified on source are not transferred to the target.

Yet another approach is to set file to immutable on the source once the file has been created. This prevents modifications of the file on the source, so it can be migrated at any time after it has been 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 have been migrated from being identified by the policy engine.

### 3.5.2 Async with HSM on target

In this section we primarily focus on async with HSM on the target and provide some guidance for ascp in, “Considerations for ascp” on page 49.

With HSM on the target directory, files that have been transferred by async to the target directory are eventually migrated by HSM. As long as a file is not modified on the source directory, this is not a problem. However, if a file that has been migrated on the target is modified in the source directory then it becomes a candidate for replication with the next async session. During the async session the file on the source will be compared to the file on the target and transferred because it has been modified in the source directory. This transfer will cause the file to be recalled in the target directory. If this happens with many files within an async session, this can cause recall storms on the target.

This situation cannot be prevented because the target migration policy has no idea about 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:** Only use HSM configured for the target directory in conjunction 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 initiate bulk recalls for these file on the target prior to 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 have been modified on source. However, if this is possible the files modified on the source have to be identified with regard to the path and file name on source. Then the source path has to be mapped to the corresponding target path. With the resulting file list bulk recalls can be executed on the target as shown in 3.5.1, “Async with HSM on source” on page 45.

One way to prevent modification for files in the source directory the IBM Spectrum Scale immutability feature can be used[14]. With this feature file on the source can be set to
immutable. An immutable file cannot be modified, renamed or deleted. Async is able to
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 have been set to
immutable. This assures that file will not change on the source and hence prevent recalls on
the target.

Example 3-27 shows an HSM MIGRATE policy on the target that only migrates file that are
immutable and that have been modified 10 days ago or longer:

\[/* 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 impacts caused by async in conjunction with
HSM configured on the target directory is to assure that files, once transferred to the target,
are not modified in the source directory again. Otherwise modifications on files on the source
cause recalls (storms) on the target. One practical approach is to set file to immutable on the
source, let async only transfer files that are set to immutable on the source and migrate file
that are immutable on the target. For more information about using IBM Aspera sync with
immutable files can be found in 3.6, “Integration with immutable files” on page 50.

Considerations for ascp

When using ascp and migrating files on the target using HSM the same challenges as
explained for async exist (see 3.5.1, “Async with HSM on source” on page 45). This is, if a file
is modified on source it will be identified by the policy engine and transferred to the target. If
the file on the target is migrated it will be recalled. If many files that are migrated on the target
are sent from the source recall storms will occur.

One way to avoid recall storms using ascp is to prevent overwrites on the target using the
parameter:

--overwrite=never
However this will cause 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 once the file has been created. This prevents modifications of the file on the source, so it can be migrated at any time after it has been set to immutable.

In summary, when using ascp in conjunction with HSM configured on the target directory, files once stored on the source should not be changed again. This can be enforced by setting files to immutable [14]. Another way is to not overwrite existing files on the target. However, this can cause inconsistencies between the source and target directory contents.

### 3.6 Integration with immutable files

IBM Aspera sync and ascp both have the capabilities to preserve the extended attributes of files transferred from the source to the target. This also includes immutability attributes that are provided by IBM Spectrum Scale [14]. Example 3-28 shows some extended attributes of an immutable file including the immutability attributes (immutable, appendOnly, indefiniteRetention and expiration Time):

```bash
# 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 README
Encrypted: no
```

The attributes immutable, appendOnly, indefiniteRetention, and expiration time shown in Example 3-28 can be set on a per file basis within an immutable fileset [14]. IBM Aspera sync preserves immutability attributes for files transferred to the target directory. This however requires that the target directory has the same fileset structure as the source directory. In particular the immutable filesets 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, 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 have changed (see 2.2, “How async works” on page 10). In this case it will only transfer and apply the changed attributes on the target file. Ascp copies any file entirely that matches the file and directory pattern given with the ascp command, including the updated file attributes. Certain limitations apply with IBM Aspera sync version 3.9 with regard to the transfer of immutability attributes:
If an immutable file has been transferred to the target and subsequently an extended attribute, such as *expiration time* or the *indefiniteRetention* flag for the file on the source are changed then async will apply 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 async to 000. For more details about setting the default file creation mask. See “File creation mask”.

If an immutable files has been transferred to the target, any subsequent transfer using ascp will always present an error message. This is because ascp attempts to transfer the file entirely and since it is immutable on the target the error message occurs. If only the expiration time or the *indefiniteRetention* flag for the file on the source is changed then ascp will apply this change to the target file, however, the error message persists.

If a file with the *indefiniteRetention* flag set to yes has been transferred to the target and subsequently 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. This issue is under investigation.

If a file with the *appendOnly* flag has been transferred to the target and subsequently this file is changed on the source then the next transfer of the changed file to the target will fail. This is because async and ascp do not support append-only mode.

Immutable filesets configured in compliant mode do not allow to unset the immutable flag for a file. However normal filesets may allow this. IBM Aspera sync (async and ascp) cannot unset the *immutable* flag on the target, hence the transfer will fail.

**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. Furthermore Async does not unset the *indefiniteRetention* flag for files on the target when it has previously been transferred from the source file. When the expiration time or the *indefiniteRetention* flag of a file changes on the source that has been previously been transferred to the target then the new expiration time or *indefiniteRetention* flag is applied upon the next run, but it will cause 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. In order to prevent this error message the default file creation mask used by IBM Aspera sync can be set to 000. With 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 shown in Example 3-29 identifies files that have been 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 may cause error if 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 have been set to immutable and the expiration time as well as indefiniteRetention flag has been set. This way it is assured 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 created in the source file system or fileset that includes the source directory. The main advantage of using 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. Hence, accessed or locked files do not impact the replication session. In addition, for applications that require a consistent recovery point, like databases, the application can create an application consistent snapshot. This snapshot can be used as source for the replication and the data that has been replicated to the target is also consistent from an application perspective.

Async does not create snapshots. The snapshot has to be created manually using the IBM Spectrum Scale command:

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

The first parameter `fsname` is the name of the file system 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 then the snapshot is created for the fileset only and resides within this fileset.

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

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

When using snapshots then the source directory for the async session is the associated snapshot directory. The source directory name must always be the same for a given replication relation and async session name. The name of the snapshot is part of the snapshot directory and hence 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 given replication relation. This can be achieved by creating the snapshot, running async and upon completion of the async session deleting the snapshot.

An alternative method is to allow creating snapshots with different names for subsequent async sessions with the same session name and using symbolic links to link the snapshot
directory with a different name to a snapshot directory with a stable name. This method however 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 creating a snapshot of the file system `/fs1` with the name `sync_fs1_snap`, the snapshot directory `/fs/.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 (see 2.2.2, “The Sync Database” on page 12). Because the source directory is the snapshot directory which is read-only, the path for the local Sync database must be explicitly set to a path within the source directory. The following async parameter can be used for this:

```
--local-db-dir=ldbdir
```

**Note:** The target directory cannot be a snapshot directory.

The target cannot be a snapshot directory. In order to create snapshots on the target after successfully completing the async session the snapshot has to be created manually using the command shown above.

To run async 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`:

```
# mmcrsnapshot fs1 sync_fs1_snap
```

The snapshot directory name that has to be used as source directory name is:

```
/fs1/.snapshots/sync_fs1_snap.
```

To run async from the snapshot of the source directory use the following command shown in Example 3-30:

```
Example 3-30   Run 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]
```

Keep in mind that the session name given with parameter `-N` and the source directory name residing on the snapshot and given with the parameter `-d` and must remain the same for all
subsequent async sessions. The later requires that the snapshot name is kept the same for all subsequent async sessions.

If files have been renamed and deleted in the source directory then the next async session from a new snapshot will propagate these changes to the target directory.

After the async session has completed, delete the snapshot so it can be recreated right before the next async session starts:

```
# mmdelsnapshot 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. This requires that the snapshot is manually created whereby the snapshot name for a given replication relation and async session name must remain the same for all subsequent async session. Snapshots can also be manually created on the target once the async session has finished.

### 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.

In order to transfer files from a snapshot of the source directory using ascp the snapshot has to be created first using the following command:

```
# mmcrsnapshot fs1 ascp_snap
```

If the source directory is `/fs1` the files residing in the source directory can be found in the following snapshot directory: `/fs1/.snapshots/ascp_snap`.

The command 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 Transfer files from a snapshot directory on the source to the target directory**

```
# ascp -l 1G -d -p --delete-before-transfer --preserve-xattr=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 have been renamed or deleted in the source directory since the last snapshot are renamed and deleted in the target directory.

Once the file transfer using the ascp command above has successfully been completed the snapshot can be deleted using the following command:

```
# mmdelsnapshot 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 then the next time an ascp transfer from a snapshot is performed the new snapshot must have a new name. 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 has to be done manually 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 details about using 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 has to be adjusted in order to remove the `.snapshots` 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 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) has been 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.

To run the policy the `mmapplypolicy` command is used. In order to scan the snapshot the snapshot path including the source directory must be given to this command 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]
```
This command runs the policy engine and creates a file list including the file names matching
the third rule of the policy above. 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 40). When using ascp the parameter --src-base must reflect the snapshot
directory (in this example /fs1/.snapshots).
Summary

In the previous chapters we have presented different options to run and integrate IBM Aspera async with IBM Spectrum Scale. In this chapter we provide some quick guidance for using these options and highlight some use cases.
4.1 Async or ascp standalone

When using IBM Aspera sync 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 discussed earlier.

If the duration to identify the files to be transferred become long using async, then 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 using IBM Aspera sync integrated in the IBM Spectrum Scale policy engine async or ascp can be used in the backend to transfer the files identified by the policy engine. We recommend using async. The advantage of using async is that it tracks files that have been transferred. This is the foundation to run full synchronizations without copying files that have been transferred already. Running full synchronizations of the source and target directory assures that all files are transferred, just in case files have not been transferred. This could occur if the network connection failed during a transfer or if the policy rules were not effective. Another advantage is that async provides more control for file transfer sessions and threads.

Using ascp independent of async integrated with the IBM Spectrum Scale policy engine has less overhead because it does not maintain the Sync Database. This is also a disadvantage because ascp copies a file even if the source and the target directory hold the same version of the file already. Running full synchronizations using async will copy all files again because there is no Sync Database for files copied with ascp.

4.3 Standalone LIST policies or external policy script

When integrating async or ascp with the IBM Spectrum Scale policy engine we described two options (3.4.1, “IBM Spectrum Scale policy engine” on page 31):

- Using standalone LIST policies
- Using external scripts

We recommend using the integration with external scripts. It is simpler from an operational perspective because it only requires one step. However, the external script has to be implemented, tested and maintained.

Using standalone 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 generated by the LIST policy and use customized transfer options in accordance to the file list.

4.4 Using snapshots

Using 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 AFM DR snapshots are not created automatically by IBM Aspera sync. This means additional 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 could 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 detail, we just want to provide some 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, then 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 facilitates the migration of files across different IBM Spectrum Scale file system levels.

Using 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 very 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 will also prevent 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 are long. The goal of disaster recovery is to have the identical content on the source and the target file system at a certain point of time.

For the disaster recovery use case async can be used standalone. This guarantees that all files are replicated and takes care of deleted and renamed files. If the duration of file identification become to 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 invoked by the policy engine. To assure that the content on source and target are identical it is recommended to perform periodic full synchronizations by running async standalone.
periodically. This takes care for files that have been missed for transfer due to network or 
policy issues. For example, if the policy to transfer files is executed every 4 hours it might be 
appropriate to run a full synchronization using async standalone once a day or once a week. 
During the full synchronization runs async standalone will crawl the file system, so it may take 
time.

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

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

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *IBM Spectrum Scale Immutability Introduction, Configuration Guidance, and Use Cases*, REDP-5507

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- [1] IBM Spectrum Scale overview

- [2] IBM Spectrum Scale Application Programming Interface documentation
  https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/bl1adm_intrfce.htm

- [3] Spectrum Scale Active File Management for Disaster Recovery
  https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/bl1ins_activefilemanagement.htm


- [5] Reference for the open source tool rsync:
  https://rsync.samba.org

- [6] Implementation of rsync for IBM Spectrum Scale
  https://github.com/gpfsug/gpfsug-tools/tree/master/bin/rsync

  https://download.asperasoft.com/download/docs/entsrv/3.9.0/cs_admin_linux/webhelp/index.html#dita/sync.html
Integration of IBM Aspera Sync with IBM Spectrum Scale

[8] Include and Exclude rules with async
https://download.asperasoft.com/download/docs/entsrv/3.9.0/cs_admin_linux/webhelp/index.html#dita-sync/filtering_rules.html

https://download.asperasoft.com/download/docs/entsrv/3.9.0/cs_admin_linux/webhelp/index.html#dita/ascp_2.html

[10] 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/b1ladm_tuningguide.htm

[11] The mmapplypolicy command:
https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b1ladm_mmapplypolicy.htm

[12] Spectrum Scale policies
https://www.ibm.com/support/knowledgecenter/STXKQY_5.0.1/com.ibm.spectrum.scale.v5r01.doc/b1ladv_policies.htm

[13] Practical guide for using IBM Spectrum Scale policies
http://w3.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102642

[14] Spectrum Scale immutability Redpaper
http://www.redbooks.ibm.com/abstracts/redp5507.html

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services