Parallelized Data Replication of Multi-Petabyte Storage Systems

Honwai Leong†
DDN Australia Ptd Ltd
Sydney, NSW, Australia
hleong@ddn.com

Andrew Janke
Information and Communications (ICT)
The University of Sydney
Sydney, NSW, Australia
andrew.janke@sydney.edu.au

Daniel Richards
DDN Australia Pty Ltd
Wellington, New Zealand
dangr@ddn.com

Stephen Kolmann
Information and Communications (ICT)
The University of Sydney
Sydney, NSW, Australia
stephen.kolmann@sydney.edu.au

ABSTRACT

This paper presents the architecture of a highly parallelized data replication workflow implemented at The University of Sydney that forms the disaster recovery strategy for two 8-petabyte research data storage systems at the University. The solution leverages DDN's GRIDScaler appliances, the information lifecycle management feature of the IBM Spectrum Scale File System, *rsync*, GNU Parallel and the MPI *dsync* tool from mpiFileUtils. It achieves high performance asynchronous data replication between two storage systems at sites 40km apart. In this paper, the methodology, performance benchmarks, technical challenges encountered and fine-tuning improvements in the implementation are presented.

CCS CONCEPTS

• Computer systems organization → Architectures → Parallel architectures → Single instruction, multiple data

KEYWORDS

Disaster recovery, data transfer, parallel, MPI, mpifileutils, Spectrum Scale, rsync, replication

ACM Reference format:

Honwai Leong, Andrew Janke, Daniel Richards and Stephen Kolmann. 2020. Parallelized Data Replication of Multi-Petabyte Storage Systems. In *Proceedings of HPC Systems Professionals Workshop 2020. ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

† Primary correspondence

HPCSYSPROS20, November 2020, Atlanta, Georgia, USA

https://doi.org/10.1145/1234567890

1 Introduction

The University of Sydney utilizes a central storage platform for research data that is managed by ICT. The platform is utilized as both traditional file shares and on local HPC facilities. The system is used by thousands of academics daily, has disaster recovery (DR) and snapshot capability, and the size continues to grow. As a result of recent growth, the university installed two 8-petabyte storage systems into datacenters separated by approximately 40km. These systems formed part of the University's technology refresh roadmap to replace existing research data storage systems. The storage systems, commonly termed as the NextGen Research Data Storage (RDS), are backed by DataDirect Networks (DDN) GRIDScaler file storage appliances. They are a tightly integrated storage platform based on DDN Storage Fusion Array (SFA) architecture and IBM Spectrum Scale File System a.k.a. General Parallel File System (GPFS). Each installation site consists of a SFA GS18K [1] dual-controller storage system with ten SS9012 disk enclosures that provide low latency high performance data throughput and over ten petabytes of raw capacity. GPFS [2], a high-performance parallel file system with rich enterprise features, is embedded into the storage controller couplet hardware as virtual machines to serve the file system, no additional hardware is required to serve as GPFS NSD servers. The storage can scale up to twenty SS9012 disk enclosures, for a total of 1,872 drives (inclusive of 72 SAS/NVMe drives in the controller enclosure) per pair of GS18K controller couplet, and/or scale out with additional SFA18K+SS9012 subsystems.

Figure 1 shows the high-level architecture of NextGen RDS installed at The University of Sydney. In addition to the GS18K with embedded GPFS NSD servers, three nodes, labeled as Cluster Export Service (CES) servers, are added to the GPFS cluster at each site. These nodes mount the file system natively as NSD clients, and export/share the file system to other non-GPFS-native clients through Network File System (NFS) and Server Message Block (SMB) protocols. NFS and SMB are currently the predominant

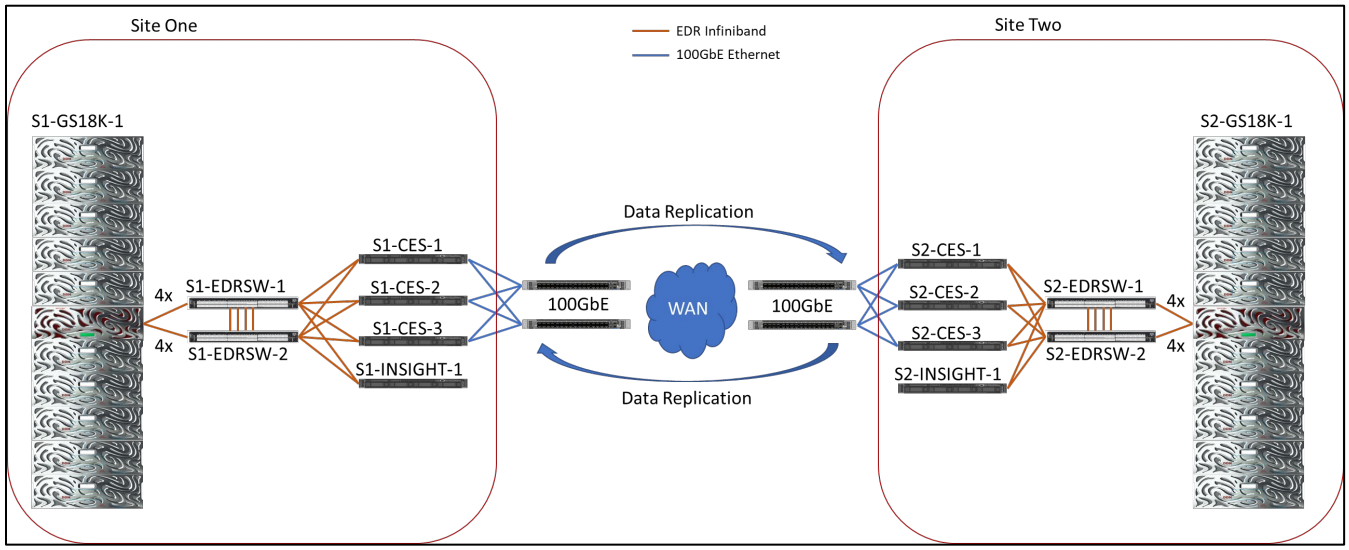

Figure 1: High Level Architecture of Next-Gen RDS implemented at The University of Sydney.

access points for end users. NextGen RDS is designed as an active-active file storage system. The storage system at each site serves distinct sets of data to end users while also being a disaster recovery (DR) site for the other.

Replication of data across sites was the grand challenge tackled in the implementation of NextGen RDS. With limited resources available, the three CES servers at each site are used to carry out the data replication. These three CES servers at each site communicate to each other across sites via a dual-link 100-GbE interconnect over the University wide area network. The backend interconnect within a site between the CES servers and the embedded NSD servers is a private EDR Infiniband network. The 100-GbE network is a routed layer three network and is the same network used by end users to access RDS via the NFS and SMB protocols.

The University has set a Recovery Point Objective (RPO) of four hours for out-of-synchronization data replication between sites, i.e. up to four hours of data loss is acceptable if one of the sites has an unscheduled outage. In a one site outage scenario, all data would be served from the surviving site. Data will be re-synchronized and replicated from the surviving site to the outage site when it recovers.

There is potential for massive number of files to be changed in any period of time, in the range of few hundred thousand of files per hour. This generates a large amount of data to be replicated across sites. Using the single-threaded *rsync* [3] [4] tool to scan and synchronize the whole file system across sites is not a feasible solution to meet the four-hour RPO. An intelligent and highly parallelized data transfer methodology is needed to achieve the RPO goal. To address this challenge, a multi-phase data replication workflow that parallelizes data replication has been developed and implemented for NextGen RDS.

The detailed methodology of the data replication workflow is discussed in section 2 of this paper; section 3 presents the benchmark performance of file transfer tools; technical challenges encountered during the implementation and fine-tuning are discussed in section 4; followed by a conclusion and discussion of future work in section 5.

2 Methodology

The following tools were explored and studied for this implementation: *rsync* [3], mpiFileUtils [4] *dsync* [6], and GNU Parallel [6]. After detailed testing and analysis, GNU Parallel with *rsync* is chosen as the main engine to drive the replication, followed by mpiFileUtils *dsync* as a failsafe to ensure data is correctly replicated from the source to the destination. Figure 2 illustrates the architectural design of the implementation.

As seen in Figure 2, the file system at each GPFS cluster is named “fs0”, mounted at /gpfs/fs0. Under the root file system “fs0”, two independent filesets with own inode space are created: one for Production and one for DR. At S1, the production fileset is named “fs-1” and the DR fileset is named “fs-2-dr”; at S2, the production fileset is named “fs-2” and the DR fileset is named “fs-1-dr”. As implied by the name of the fileset itself, “fs-1-dr” at S2 serves as the DR copy of Production data stored under “fs-1” fileset at S1; “fs-2-dr” at S1 serves as the DR copy of Production data stored under “fs-2” fileset at S2. Under each independent fileset, unique research project directories are created for each individual research project group (the end users). Each project directory is named the same under respective production and DR filesets, e.g., if there is a project directory named “PROJ-ABC” under “fs-1” Production fileset at S1, there exists an equivalent project directory named “PROJ-ABC” under “fs-1-dr” DR fileset at S2. With replication in place, these directories contain the same datasets. While both Production and DR filesets are exported/shared through NFS and SMB protocols at all time, the datasets in the DR filesets are only accessible in read-only mode during normal operation time.

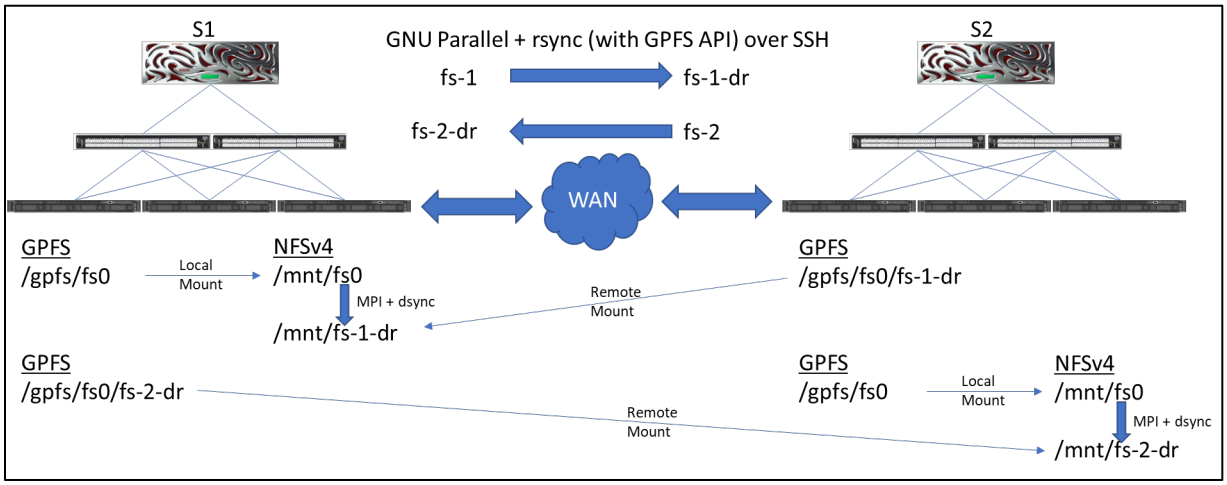

Figure 2: Detail architecture design of data replication from S1 to S2 and vice versa.

The overall replication workflow consists of the following phases:

1. Create a global snapshot of the file system at the local site.
2. Compare latest and previous snapshots to generate lists of file/directory changes.
3. Utilize GNU Parallel with *rsync* to replicate files and directories in the generated list (from phase 2) to the remote site.
4. Utilize mpiFileUtils *dsync* to perform a complete synchronization from the Production fileset to DR fileset.

The workflow is initiated by the standard Linux *cron* utility running on one of the CES nodes at each site. While the RPO is four hours, the *cron* job is triggered every hour. A lock file is used to prevent a new workflow cycle from starting if the previous cycle is still in progress.

2.1 Global File System Snapshot

The first phase of the workflow is to create a global snapshot of the file system at the local site using the built-in GPFS snapshot tool. A GPFS snapshot provides a point-in-time persistent image of the file system. All files and directories in a snapshot are static. Replicating data from the snapshot avoids the complication of dealing with live data (e.g. files could be modified or deleted during replication process). Using snapshots also eases tracking the differences between the Production and DR filesets. Snapshot files and directories are accessible from the ".snapshots" directory in the root directory of the file system, for example "/gpfs/fs0/.snapshots/${snapshot}", where "${snapshot}" is the unique name of a snapshot.

2.2 Information Lifecycle Management

Leveraging on GPFS' Information Lifecycle Management (ILM) fast scan policy engine, tracking changes in the file system is a simple low-cost operation. The "MODIFICATION_SNAPID" attribute in ILM provides a smart mechanism to track the file changes after a specific file system snapshot is taken. Snippet 1 defines the policy rule used in ILM to find changes made since a specific snapshot was taken.

```
RULE 'new' LIST newlist
    DIRECTORIES_PLUS
    WHERE MODIFICATION_SNAPID >= prevsnapid    (1)
```

In snippet 1, "prevsnapid" refers to a specific snapshot point of the file system. Files and directories that are newly created or modified after this snapshot was taken would be listed as eligible candidates for further execution by the policy engine, as defined in snippet 2. "replication-exec.sh" is a user defined script that performs actions on the lists of eligible candidates.

```
RULE EXTERNAL LIST newlist exec \
'/path/to/replication-exec.sh'                (2)
```

As seen in snippet 3, *mmapplypolicy* is the GPFS command used to launch the ILM policy engine and filter files based on rules written in the "replication.policy" file which consists of rules defined in snippets 1 and 2. When processing this policy, the policy engine scans and searches the production fileset in the most recent snapshot, "/gpfs/fs0/.snapshots/${lastsnapshot}/${prod_fileset}/", with "prevsnapid" referencing the previous snapshot. This policy is a comparison for file changes between the most recent and previous snapshots of the file system. Files/directories in the most recent snapshot which were newly added or modified since the previous snapshot would be listed as candidates for further execution by ILM.

The *mmapplypolicy* command provides the flexibility to select nodes (defined by the argument of *-N* option) and the number of helper threads per node (defined by argument of *-m* option) to participate in the scan and execution phases of the policy. Additional nodes and threads distribute the workload to reduce the overall time-to-completion. In this implementation, all three CES nodes at each site are selected with three helper threads per node.

```
mmapplypolicy \
/gpfs/fs0/.snapshots/${lastsnapshot}/${prod_fileset}/ \
-P replication.policy -N ces1,ces2,ces3 -m 3 \
-M prevsnapid="SNAPID(${prevsnapshot})"                    (3)
```

2.3 GNU Parallel with *rsync*

ILM generates multiple lists of files and directories that are candidates for dispatch by the user defined script i.e. "replication-exec.sh" in snippet 2. The script launches GNU Parallel with *rsync* to copy the listed files and directories from the local Production fileset to the remote DR fileset over SSH, see snippet 4.

```
parallel -j 4 rsync -RltgoWd \
--gpfs-attrs --rsync-path='/path/to/rsync --gpfs-attrs' \
/gpfs/fs0/.snapshots/${snapshot}/${prod_fileset}/./{} \
${remotehost}:/gpfs/fs0/${dr_fileset}/ :::: ${list}        (4)
```

As seen in snippet 4, data replication is accomplished by the *rsync* [3] [4] file replication tool. The *parallel* binary wrapper in snippet 4 parallelizes the execution by spawning multiple *rsync* processes (the number of processes is set by the integer argument passed to *-j* option). Files and directories in "${list}" (generated by ILM) are distributed to *rsync* threads started by *parallel*. Together with the *mmapplypolicy* multi-threading, GNU Parallel further compounds the scale of the parallelization with each *mmapplypolicy* helper thread spawning multiple *rsync* processes. Figure 3 shows an example of how the parallelization is achieved.

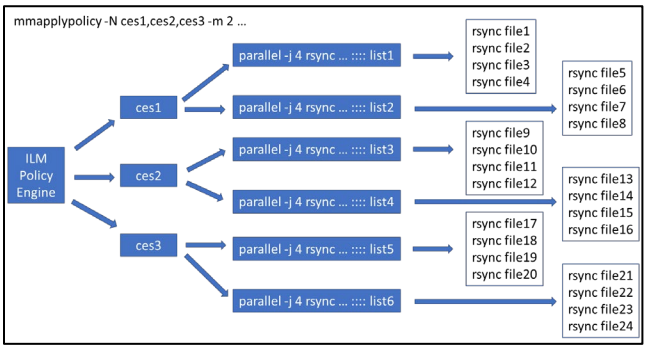

Figure 3: *mmapplypolicy* launches the ILM policy engine using *-N ces1,ces2,ces3* (three nodes) and *-m 2* (two helper threads per node). Each helper thread executes *parallel* with *-j 4* which spawns four *rsync* processes. Altogether 24 *rsync* processes replicate 24 files concurrently.

The *rsync* binary used in this implementation is a modified version of *rsync* that also synchronizes the extended attributes of files/directories used in GPFS with the *--gpfs-attrs* option. GPFS APIs are added into the source code of *rsync* version 3.1.3, based on the modification made in reference [4].

2.4 mpiFileUtils *dsync*

After completion of the ILM and parallel *rsync* phases, mpiFileUtils [8] *dsync* [6] is next in the replication workflow to synchronize the local Production fileset and the remote DR fileset. MpiFileUtils *dsync* is an opensource Message Passing Interface (MPI) application that synchronizes two files or two directory trees so that source and destination will have identical content, ownership, timestamps and permissions. The use of *dsync* serves two purposes: to delete files in the DR fileset that have been deleted from the source Production fileset; and to act as a failsafe to cover files that are missed during replication in previous ILM and parallel *rsync* phases (due to the use of some special characters in the file path not interpretable by ILM). Open MPI [9] implementation is used as the MPI wrapper for *dsync* in this workflow.

Though *rsync* is also capable of deleting files at the destination during a synchronization between two directories, recursive *rsync* is a very time-consuming process. ILM is only able to provide lists of new and updated files/directories between two snapshots that need to be replicated over to DR fileset, it is not able to identify deleted files between two snapshots. Thus, *dsync* is needed here to cover the gap.

Furthermore, due to the use of special characters and spaces by end users in their file paths, ILM intermittently fails to copy these files to DR site. ILM also does not list renamed files or directories as eligible candidates. Thus, the use of *dsync* provides a failsafe measure to ensure all files and directories in Production and DR filesets are consistent.

As seen in Figure 2, at each site, *dsync* is executed on NFS mount points of local GPFS (local /gpfs/fs0/ is NFS exported and mounted locally as /mnt/fs0) and remote GPFS DR fileset (remote /gpfs/fs0/fs-1-dr or /gpfs/fs0/fs-2-dr is NFS exported and mounted locally as /mnt/fs-1-dr or /mnt/fs-2-dr respectively at each site). Unlike *rsync*, which could copy files from a local server to another remote server over SSH communication, *dsync* is limited to operations on a local server only. To maintain GPFS extended attributes, *dsync* is restricted to work on mount points of the same type, e.g. from GPFS to GPFS or from NFS to NFS, and not from NFS to GPFS nor vice versa. Static files are copied from /mnt/fs0/.snapsots/${snapshot}/${prod_fileset} directory to corresponding remote DR fileset directory (i.e. /mnt/fs-1-dr or /mnt/fs-2-dr) using MPI *dsync* over NFS (see snippet 5). Similar to previous phase, all three CES nodes at each site participate in the MPI *dsync* phase.

```
mpirun -allow-run-as-root -n 24
-N 8 -host ces1:8,ces2:8,ces3:8 -map-by socket \
-bind-to core dsync -D -s \
/mnt/fs0/.snapshots/${snapshot}/${prod_fileset}/${project} \
/mnt/{dr_fileset}/${project}                               (5)
```

3 Performance Benchmark

The design of the replication workflow presented in section 2 is guided by performance benchmark of parallel *rsync* and MPI *dsync*. Comparing how *rsync* and *dsync* function in this workflow, one may comment that ILM and parallel *rsync* are redundant steps since *rsync* is limited to copying only files listed as candidates by ILM and is not used to delete removed files (from source) at remote DR fileset, while *dsync* alone could accomplish both tasks. The inclusion of parallel *rsync* into the workflow is driven by the

superior scalability of parallel *rsync* over MPI *dsync* in terms of aggregate data transfer bandwidth performance.

Benchmark performance tests were carried out to measure the performance of parallel *rsync* and MPI *dsync*. The benchmark is setup to copy multiple 4 GiB files from GPFS at the local site to GPFS at the remote site. Table 1 and Figure 4 show the performance benchmark of parallel *rsync* scaling from a single process on a single node up to 24 processes per node on three nodes; Table 2 and Figure 5 show the performance benchmark of MPI *dsync* from a single process on a single node up to sixteen processes per node on three nodes. Comparing one-to-one process, *rsync* outperforms *dsync* with double the data transfer bandwidth (~156 MiB/s vs. ~71 MiB/s). Parallel *rsync* is also shown to be more scalable than MPI *dsync* with up to ~4.3 GiB/s on three nodes with 24 *rsync* processes per node, while MPI *dsync* reaches the upper limit of ~560 MiB/s on three nodes with eight *dsync* processes per node. The slower performance of MPI *dsync* is suspected to be limited by NFS, whereas parallel *rsync* reads and writes directly to GPFS. As explained in section 2.4, *dsync* is restricted to work on NFS-to-NFS in order to maintain the GPFS extended attributes of files/directories.

Whether using ILM together with parallel *rsync*, or MPI *dsync* alone, a scan through the file system is required: ILM scans the differences between two snapshots taken at different time; MPI *dsync* scans the difference between two NFS directories (one local and one remote). ILM is capable of scanning about 450,000 files per second using three nodes with three helper threads per node; using three nodes and eight processes per node, MPI *dsync* could only scan up to about 30,000 files per second on locally NFS exported production fileset and up to about 20,000 files per second on remotely NFS exported DR fileset (slower due to further distance and higher latency in a layer-3 network).

Based on these benchmark results, the replication workflow is designed to leverage on the better performance of ILM to scan for lists of new and updated files in production fileset, and use the more scalable parallel *rsync* to copy them to the remote DR fileset, followed by MPI *dsync* for deletion of removed files (from the Production fileset) at the remote DR fileset, and acts as a failsafe to ensure both the Production and DR filesets are in sync.

4 Technical Challenges and Fine-Tuning Improvements

Some initial technical challenges were encountered when the replication workflow was placed into live production. Fine tunings have been carried out in the workflow to address these challenges.

Table 1: Performance Benchmark of parallel *rsync*.

Processes per Node	Bandwidth Performance (MiB/s)		
	1-node	2-node	3-node
1	156.372	307.415	471.039
2	309.951	606.950	916.194
4	592.764	1,146.857	1,775.081
8	1,123.423	2,191.546	3,347.431
16	1,397.207	2,707.484	4,110.386
24	1,523.172	2,985.967	4,408.910

Table 2: Performance Benchmark of MPI *dsync*.

Processes per Node	Bandwidth Performance (MiB/s)		
	1-node	2-node	3-node
1	71.147	144.495	212.239
2	154.905	311.270	420.354
4	193.012	368.532	554.356
8	196.661	375.457	563.938
16	196.940	376.286	560.813

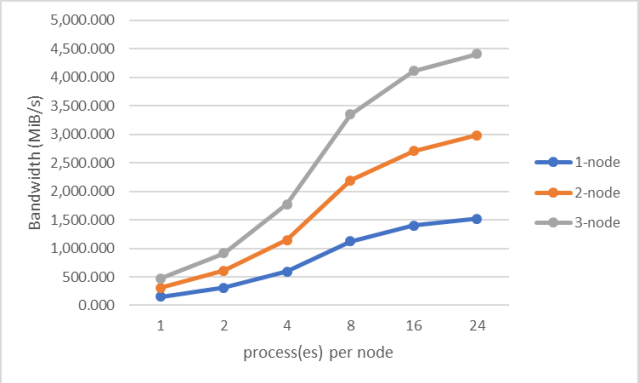

Figure 4: Performance Benchmark of parallel *rsync*.

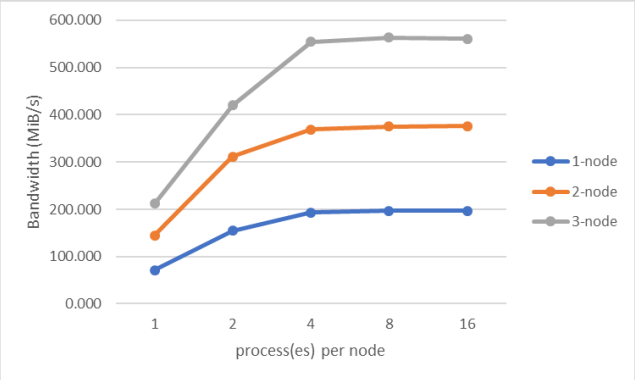

Figure 5: Performance Benchmark of MPI *dsync*.

4.1 Replication of Large Files

The scalable effect of parallel *rsync* is significant only when there are large numbers of files to be copied. When there is a mix of large and small files in the lists generated by ILM, it is not uncommon to see one or two *rsync* processes still copying a few very large files while the rest of the *rsync* processes finish copying small files. Hence large files tend to slow down the overall efficiency (i.e. longer time-to-completion) of the replication workflow. As file size increases efficiency decreases, e.g. few hundred Gigabytes per file.

With parallel *rsync*, even though GNU Parallel could spawn multiple *rsync* processes concurrently, it is a one-to-one mapping where each *rsync* process can only copy a single file at a time. If there is only one file to be copied, GNU parallel would only start a single *rsync* process to copy the file, regardless of what *-j* value is passed to the *parallel* wrapper. Where, MPI *dsync* processes could decompose a large file and all *dsync* processes participate in copying the file in parallel (even though it is only one file), thus accelerating the data transfer. For a single large file, the data transfer rate using *rsync* remains at 155 MiB/s while MPI *dsync* could potentially reach 560 MiB/s.

To avoid the overhead of large file copy using the slower *rsync*, the original ILM policy (see snippet 1) is changed to exclude large files as candidates for parallel *rsync* (see snippet 6). Excluded large files are left to be copied by MPI *dsync*, leveraging the better data transfer rate of MPI *dsync*, improving the overall workflow efficiency. In the current implementation, files larger than 200 GiB in size are excluded from parallel *rsync*. It is noted that this change may introduce negative impact to the workflow if there are large numbers of 200 GiB files to be copied. In this instance the aggregate bandwidth of parallel *rsync* copying multiple files would work more efficiently than using MPI *dsync*. The negative effect is difficult to measure as each cycle of the ILM scans list different number of large files and they are in different sizes. It is still a work in progress to determine the optimal file size threshold that should be set to achieve a balance between when to use parallel *rsync* and when to use MPI *dsync*.

```
RULE 'new' LIST newlist
    DIRECTORIES_PLUS
    WHERE MODIFICATION_SNAPID >= prevsnapid
    AND FILE_SIZE < 214748364800                    (6)
```

4.2 Memory Caching

mpiFileUtils *dsync* is a memory intensive application, especially when copying large files. It leads to heavy memory caching effect on the servers where the processes run. Due to limited resources available, CES nodes, the same resource group used for NFS and SMB access by end users, are used to execute MPI *dsync*. Due to the heavy memory caching effect of MPI *dsync*, it has caused undesired interruptions to NFS and SMB services.

The original source code of mpiFileUtils *dsync* does not have an option to use direct I/O, i.e. bypassing the caching layer and write/read directly to-and-fro backend server disks. To mitigate the default memory intensive nature of *dsync*, the source code of *dsync* is modified to allow the use of direct I/O. A new *-s* option is added to *dsync* to enable direct I/O (see snippet 5). This change is proven crucial in eliminating the heavy memory caching effect caused by *dsync*, thus preventing it from interrupting NFS and SMB services. This has been at the cost of sacrificing some performance with MPI *dsync* (due to lack of cache assistance).

4.3 Massive number of files and directories

Like any other storage systems deployed for academic research, The University of Sydney stores massive numbers of files and directories in RDS. Research data is stored in different research projects, where each project has its own project directory under either the “fs-1” or “fs-2” Production filesets (and the corresponding “fs-1-dr” or “fs-2-dr” DR filesets). The first iteration of the replication workflow is to have MPI *dsync* walk through all project directories (both Production and DR) and synchronize them where necessary.

Initially when RDS went into live production, the overall replication workflow easily completed within the four-hour RPO window. As the number of files grew, it clearly becomes more challenging to keep up with the four-hour RPO window. While it is a good to have failsafe to ensure both production and DR filesets are in sync by walking through all project directories, it is noted that unless there are files being added or updated in a project directory, a walk through all untouched project directories are deemed unnecessary.

In order to reduce the cycle time spent in walking through idle project directories using MPI *dsync*, additional rules (see snippet 7) are added into the ILM policy to scan for changed directories under the Production fileset and list them separately for execution by a different user defined script (“chgprj-exec.sh”). The “chgprj-exec.sh” script filters out subdirectories and greps only the names of the changed project directories listed by ILM (that usually include absolute paths of changed project directories and subdirectories), then prints the output into a list. A *for* loop is used by MPI *dsync* to walk through only each of the project directories in the sorted list. The overall runtime of the workflow is significantly reduced with this change in place.

```
RULE EXTERNAL LIST chgprj exec \
'/path/to/chgprj-exec.sh'
RULE 'chgprj' LIST chgprj
    DIRECTORIES_PLUS
    WHERE MODIFICATION_SNAPID >= prevsnapid
    AND MODE LIKE 'd%'                              (7)
```

5 Conclusion and Future Works

A parallelized asynchronous data replication workflow has been successfully implemented at The University of Sydney between two 8-petabyte RDS systems. Depending on the number and size of files needed to be replicated between sites, the duration of each workflow cycle varies. At the time when this paper is written, average time taken to complete a cycle is around two hours for average daily research workload.

Currently, the workflow is initiated by a *cron* job running on one of the CES nodes. Though it is simple to switch the *cron* job between nodes (in case of maintenance), the switching procedure is not automated. All three CES nodes at each site are used in the replication workflow. Though there are some monitoring in place, the existing method is not sufficiently resilient to prevent a down node from interrupting the workflow. In addition, if a workflow ends early and there is still plenty of time within the hour before the next cycle starts, time is wasted when a new cycle could have started right after the last cycle ends to further improve the RPO. Some investigations are in the works to improve the resiliency of the workflow, e.g. integrating with SLURM [10] job scheduler or Jenkins [11] automation server.

Layer-3 routed 100-GbE network is currently used for communication between the two RDS systems separated by 40km. With the dual-link connection, the highest achievable aggregate data transfer bandwidth between sites is only around 20% of the theoretical bandwidth. The limit could be server bound, or network bound due to routing and congestion in the shared University wide area network. Adding more servers to the replication workflow may help to drive more bandwidth. Some ongoing considerations to improve the data transfer bandwidth between sites include adding more uplinks, as well as removing the routing layer between sites, i.e. migrating into a layer-2 directly connected network.

Finally, it is still the best practice to invest in additional hardware to segregate the replication workflow from sharing the same hardware resources used to serve NFS and SMB. This would prevent service interruption to NFS and SMB caused by replication and vice versa. Using separate hardware would also improve the overall performance of the replication (by not using Direct I/O) as well as NFS/SMB (no resource contention with replication). It might be worth investigating potential use of container technology to better confine resource sharing between NFS/SMB and replication as an alternative workaround to adding more hardware.

ACKNOWLEDGMENTS

The authors would like to extend gratitude and appreciation to all participating staff from both The University of Sydney and DDN for contributing countless hours of efforts in successfully delivering the solution presented in this paper.

REFERENCES

[1] DataDirect Networks (DDN), "SFA18K," [Online]. Available: https://www.ddn.com/products/converged-storage-platform-sfa18kx/.

[2] International Business Machine (IBM), "IBM Spectrum Scale," [Online]. Available: https://www.ibm.com/support/knowledgecenter/STXKQY/ibmspectrumscale_welcome.html.

[3] A. Tridgell, P. Mackerras and W. Davison, "rsync," [Online]. Available: https://rsync.samba.org.

[4] A. Tridgell, P. Mackerras and W. Davison, "rsync(1) - Linux man page," [Online]. Available: https://linux.die.net/man/1/rsync.

[5] GPFS Usergroup, "gpfsug-tools/bin/rsync," [Online]. Available: https://github.com/gpfsug/gpfsug-tools/tree/master/bin/rsync.

[6] LLNL/LANL/UT-Battelle/DDN, "dsync," [Online]. Available: https://mpifileutils.readthedocs.io/en/v0.10.1/dsync.1.html.

[7] O. Tange, GNU Parallel 2018, 2018.

[8] D. Sikich, G. D. Natale, M. LeGendre and A. Moody, "mpiFileUtils: A Parallel and Distributed Toolset for Managing Large Datasets," in *2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, Denver, 2017.

[9] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham and T. S. Woodall, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in *11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, 2004.

[10] SchedMD, "SLURM Workload Manager," [Online]. Available: https://slurm.schedmd.com.

[11] "Jenkins," [Online]. Available: https://www.jenkins.io.

APPENDIX

The following artifacts are attached to this paper:

main.sh - Main replication workflow.

replication.policy - Policy used by *mmapplypolicy* command.

replication-exec.sh - Execution script of listed candidates (generated by ILM) to be replicated to DR site.

chgprj-exec.sh - Execution script of listed changed project directories (generated by ILM) to be processed by MPI *dsync*.