Report on Adaptable Open-Source Disaster Recovery Solution for Multi-Petabyte Storage Systems

Honwai Leong†
The University of Sydney
Sydney, NSW, Australia
honwai.leong@sydney.edu.au

ABSTRACT

The current generation of Research Data Store (RDS) at The University of Sydney comprises a pair of peta-scale data storage systems. We implemented a disaster recovery (DR) solution for data protection against catastrophic failure at either storage system. To handle large amount of data transactions into RDS, we took an open-source approach to design an adaptable DR solution that enables parallelized data replication capability between the pair of storage systems. In the last three years of operations, the DR solution has gone through a few iterations which saw improvement in efficiency. In this paper, we present the findings and outcomes from our DR implementation.

CCS CONCEPTS

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

KEYWORDS

Disaster recovery, data transfer, parallel, data replication

ACM Reference format:

Honwai Leong. 2023. Report on Adaptable Open-Source Disaster Recovery Solution for Multi-Petabyte Storage Systems. In *Proceedings of HPC Systems Professionals Workshop 2023. ACM, New York, NY, USA, 5 pages.*

1. Introduction

The University of Sydney Research Data Store (RDS) platform commenced service in May 2020. Daily operations see the platform utilized by thousands of academics for both traditional file shares and on local HPC facilities. RDS is hosted in two storage systems installed at two different data centers separated by approximately 40km. Over the course of last three years, the total useable capacity of RDS has grown from 8 petabytes to >13 petabytes at each site. At the time of writing, both storage systems are 85% filled. Like any other storage, RDS requires data protection against catastrophic failure. The large amount of data transactions into RDS and limited resources have driven us to take a non-trivial approach in designing the data protection plan. Together with our storage partner, DDN, we implemented an adaptable open-source parallelized data replication workflow to serve as the disaster recovery (DR) solution for RDS.

We first presented this workflow in SIGHPC Systems Professionals Workshop 2020 [1]. Over the last three years of operation, the adaptability of our workflow allows us to revise it as needed based on the changing conditions of RDS workload. We present in this paper the findings from our data replication workflow over the course of last three years since its inception.

Section 2 of this paper gives a recap of the data replication workflow; section 3 presents the revisions made to the workflow in the last three years; section 4 presents the results of the workflow; followed by conclusion in section 5.

2. High-Level Architecture

Figure 1 shows the high-level architecture of RDS currently in operation at The University of Sydney. At each site, 16 enclosures of disks provide the backend 13 PB storage to the frontend servers which host a single namespace parallel file system through NFS and SMB filesharing protocols to the end users. 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 storage system. We implemented a workflow leveraging on file system snapshots, *rsync* [2] parallelized by GNU Parallel [3], and mpiFileUtils' *dsync* [4] to achieve asynchronous data replication between sites. The combination use of parallel *rsync* and MPI *dsync* is chosen for a balance performance optimization.

Full details of the implementation are presented in [1]. Summary of the workflow is given below (Figure 2):

† Primary correspondence

HPCSYSPROS23, November 2023, Denver, Colorado, USA

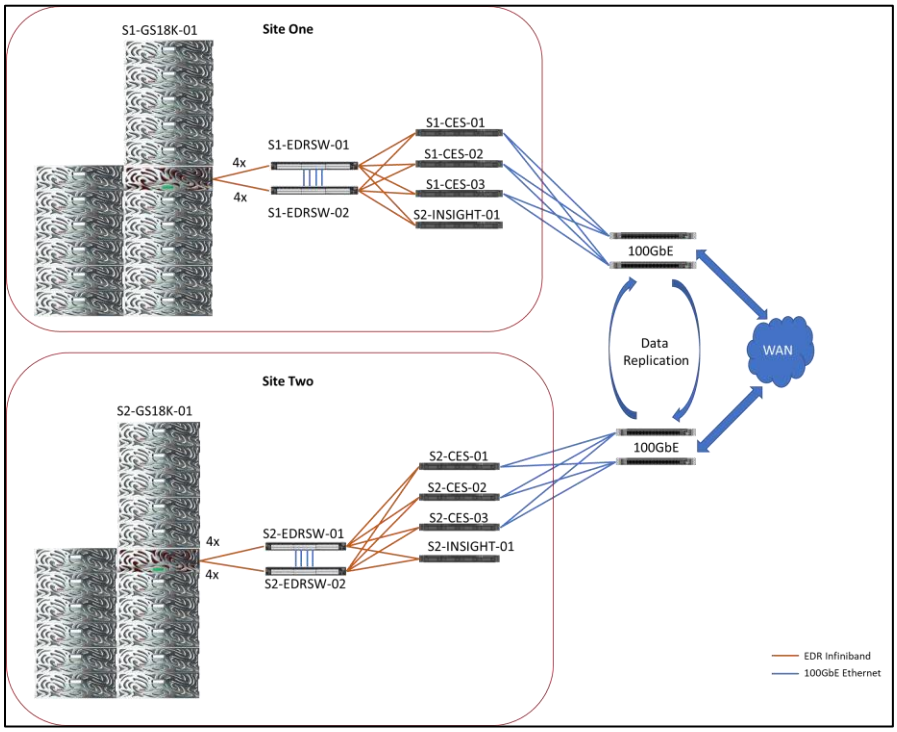

Figure 1: High-level architecture of RDS at The University of Sydney.

1. Compare file system snapshots to generate a list of new and changed files/directories that need to be replicated.
2. Candidate files/directories are replicated across site using *rsync* (non-recursive). Concurrent *rsync* of multiple files is achieved using GNU Parallel.
3. Run mpiFileUtils' *dsync* to perform a complete data synchronization of changed directories between sites, including deletion of files at destination which have been deleted at source.

The data replication workflow is scheduled to run every hour with mechanism in place to prevent overlapping run if the previous cycle does not complete within an hour. We set a four-hour RPO target for this workflow which we achieved >90% of the time. Fine tunings were made to the workflow over the last three years as needed to improve efficiency. We call our workflow adaptable and open source because it is all written in shell scripts that incorporate different open-source tools. We can edit the scripts at different sections to suit the changing conditions of RDS workload.

3. Revisions of workflow

As RDS workload increases over the years, the growing complexity of the file system prompted for fine tunings to the data replication workflow.

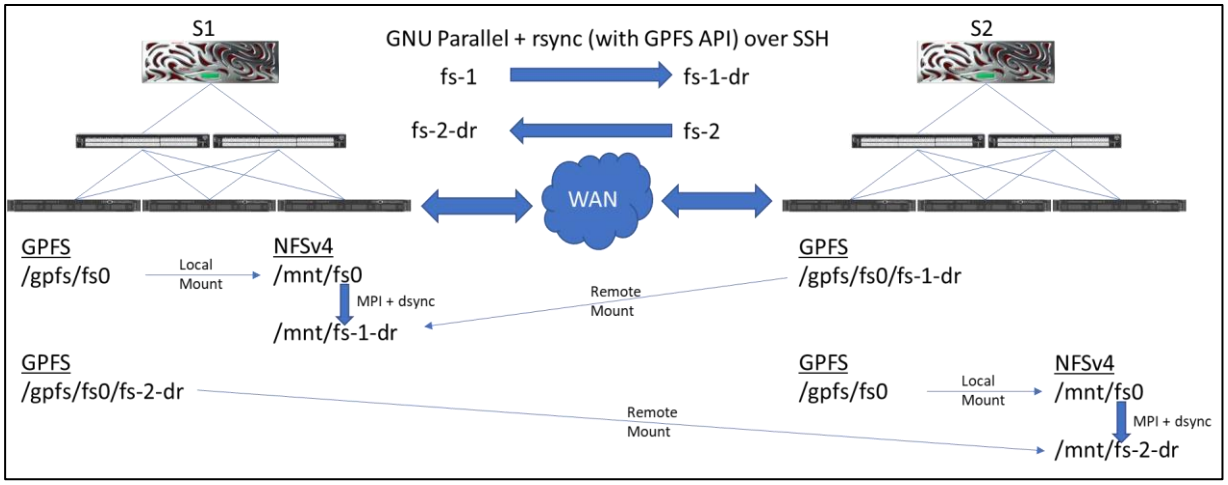

Figure 2: Illustration of data replication workflow between sites.

3.1. Optimization of *dsync* phase

As mentioned in section 2, our workflow uses *dsync* to do a complete data synchronization between sites. The hierarchical structure of RDS is organized into different project directories. In the very first iteration of our workflow, we ran *dsync* on each of the project directories, one after another to ensure 100% data consistency between sites. This allows us to track the progress of the replication and reduce the amount of memory consumed by *dsync* by compartmentalizing the file system (instead of running *dsync* on the root directory of the file system). As the number of projects grows (over 6,000 projects today), this primitive approach was proven inefficient (though reliable ensuring 100% synced between sites) as *dsync* spent time in all project directories that may or may not have any new/changed item. Thus, we modified our workflow to only run *dsync* on project directories that have new/changed items. Though this change saw improvement in reducing the replication cycle time, the workflow still occasionally exceeded the target RPO due to some large project directories with many millions of files.

To further improve efficiency, we narrowed down the workflow to run *dsync* only on sub-directories with new/changed items (Figure 3). In a project directory with changes, the change may only occur in sub-directory at any level down the project directory tree. We setup *dsync* to start on that sub-directory, by-passing its unchanged parent directories, and not rerunning *dsync* on any changed sub-sub-directory further down the starting sub-directory. We track all identified changed directories for each replication cycle to ensure our latest approach covers all of them (Figure 3).

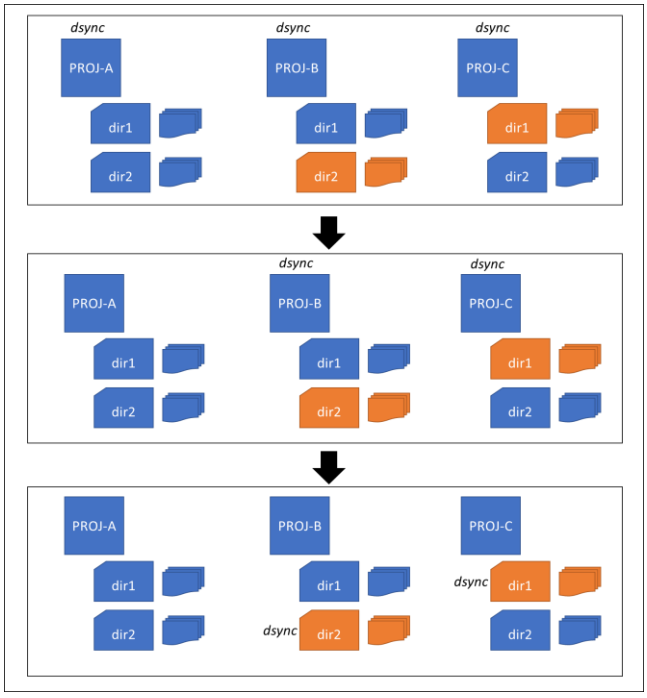

Figure 4: Optimization of *dsync* phase: (1) Initially, *dsync* is run on each project directory regardless of if the project has any new/changed item. (2) Then *dsync* is run only on project directories with new/changed item. (3) Finally, *dsync* is further narrowed down to run only on subdirectories with new/changed items.

```
[2023-07-13T21:33:26-AEST] Syncing "PROJ-A/testdir1/subdir1".
...
...<output trunctated>...
...
[2023-07-13T21:33:34-AEST] "PROJ-A/testdir1/subdir1/subdir2"
is under "PROJ-A/testdir1/subdir1" which is already synced.
```

Figure 3: Example of log showing *dsync* running on changed sub-directory in a project directory and skipping any further changed sub-sub-directory under it.

While we see further improvements to replication cycle time with this approach, we observed extra overheads incurred to the workflow. Given *dsync* is an MPI application, it induces overhead when setting up the MPI communication between data mover nodes. We had a rare situation where a project directory had changes to multiple of its sub-directories at the same level, leading to long running time caused by the extraneous MPI overheads for each subdirectory *dsync* worked on.

3.2. Exception treatment to extreme large project directories

Using the revised workflow described in section 3.1, there would be inevitable situation where the top project directory itself is identified as a changed directory. When this happens, *dsync* would walk through and compare the whole project directory tree between the source and the destination. We have a few extreme large project directories (>30 millions of files) that we want to avoid *dsync* keep working on them in every replication cycle if the top project directory is identified as changed. The longer the replication cycle, more backlog of file system changes would be incurred to the subsequent cycle(s). To mitigate this, we applied exception treatment to these extreme large directories by *dsync*-ing them only during the weekend if these top project directories are identified as changed over last one week. The parallel *rsync* phase of changes within these extreme large project directories will remain in every replication cycle to ensure new/changed data are still replicated but delaying deletion of data at the destination (due to deletion at source) and recopying of data (due to file/directory renaming at the source) to the weekend *dsync*.

3.3. Effect of cached/buffered memory

Using *dsync*, we observed that the buffered/cached memory of the data mover node(s) builds up and accumulates for each *dsync* run. The *dsync* binary calls the *sync()* function to sync data to storage. We noticed that the *sync()* function took longer to complete as the buffered/cached memory increases for successive *dsync* runs (Figure 5, Figure 6).

To optimize the *sync()* function, we put in a sanity check between *dsync* runs to drop the caches (see snippet 1) if the available free memory of the data mover node(s) falls below a configurable threshold.

```
[2020-11-30T11:20:03] Copying items to destination
[2020-11-30T11:20:03] Copying to /mnt/rds-2-dr/PRJ-DEF
[2020-11-30T11:20:03] Items: 57919
[2020-11-30T11:20:03]   Directories: 511
[2020-11-30T11:20:03]   Files: 57408
[2020-11-30T11:20:03]   Links: 0
[2020-11-30T11:20:03] Data: 43.246 GB (789.897 KB per file)
...
... <output truncated>
...
[2020-11-30T11:21:48] Copying data.
[2020-11-30T11:21:58] Copied 4.053 GB in 10.350 secs (401.039 MB/s) ...
[2020-11-30T11:22:08] Copied 7.891 GB in 20.331 secs (397.453 MB/s) ...
[2020-11-30T11:22:18] Copied 11.242 GB in 30.541 secs (376.930 MB/s) ...
[2020-11-30T11:22:28] Copied 14.240 GB in 40.512 secs (359.942 MB/s) ...
[2020-11-30T11:22:38] Copied 17.186 GB in 50.593 secs (347.845 MB/s) ...
[2020-11-30T11:22:48] Copied 20.283 GB in 60.598 secs (342.750 MB/s) ...
[2020-11-30T11:22:58] Copied 23.234 GB in 70.590 secs (337.033 MB/s) ...
[2020-11-30T11:23:08] Copied 26.753 GB in 80.627 secs (339.779 MB/s) ...
[2020-11-30T11:24:28] Copied 30.707 GB in 160.847 secs (195.492 MB/s) ...
[2020-11-30T11:24:28] Copied 43.246 GB in 160.847 secs (275.315 MB/s) done
[2020-11-30T11:24:28] Copy data: 43.246 GB (46434778579 bytes)
[2020-11-30T11:24:28] Copy rate: 275.315 MB/s (46434778579 bytes in 160.847251 seconds)
[2020-11-30T11:24:28] Syncing data to disk.
[2020-11-30T11:24:34] Sync completed in 5.663370 seconds.
```

Figure 5: Despite only a few small files being copied from source to destination, the *sync()* function call in *dsync* took more than 30 seconds to complete. This happened when the buffered/cached memory on the data mover node(s) was under high load.

```
[2020-11-17T16:46:04] Copying items to destination
[2020-11-17T16:46:04] Copying to /mnt/rds-2-dr/PROJ-ABC
[2020-11-17T16:46:04] Items: 7
[2020-11-17T16:46:04]   Directories: 0
[2020-11-17T16:46:04]   Files: 7
[2020-11-17T16:46:04]   Links: 0
[2020-11-17T16:46:04] Data: 10.362 MB (1.480 MB per file)
...
... <output truncated>
...
[2020-11-17T16:46:04] Copying data.
[2020-11-17T16:46:04] Copy data: 10.362 MB (10865754 bytes)
[2020-11-17T16:46:04] Copy rate: 77.512 MB/s (10865754 bytes in 0.133687 seconds)
[2020-11-17T16:46:04] Syncing data to disk.
[2020-11-17T16:46:35] Sync completed in 30.467195 seconds.
```

Figure 6: In another sample of copying more than 50,000 of files (total >40 GB in size), the *sync()* function call took only slightly more than 5 seconds to complete when the buffered memory on the data mover node(s) was low.

```
# sync; echo 3 > /proc/sys/vm/drop_caches          (1)
```

We also included this cache drop step at the end of the workflow to reset the data mover node(s) into a clean slate before starting a new cycle.

3.4. Dealing with Anomaly Activities

The adaptability of our workflow allows us to quickly deal with anomaly activities in RDS that affected the data replication workflow performance. By monitoring the completion time of each replication cycle, alerts were sent out to system administrators when the workflow cycle exceeded four hours. The logs generated from the workflow allow us to analyze the progress of the workflow and identify the root cause of the workflow exceeding four hours. In the case of anomaly activities, we modify our workflow to exclude replicating the project directory where the anomalies happen. Depending on the severity, we could either totally rule out the project directory from the workflow, or partially excluding it from either the *rsync* or *dsync* phase. We had a situation where a user made a mistake in their backup scripts which multiply the creation of the same sub-directories into infinite loops down the project directory tree. For such incident, the user was notified, and the project directory involved was excluded from subsequent replication cycles until the problem was rectified by the user.

4. Performance analysis of workflow

In this section, we evaluate the performance of our data replication workflow over the last three years between June 2020 to June 2023 period. We measure the performance by analyzing the time taken to complete each replication cycle. In this three-year period, there were a total of 16,077 replication cycles completed from site 1 to site 2 and 11,484 cycles completed from site 2 to site 1. The replication cycle time from site 2 to site 1 is typically longer (thus lower number of cycles) due to higher loading of data into site 2. Overall, we achieved the four-hour RPO target 96.430% of the time from site 1 to site 2 and 90.866% from site 2 to site 1.

Table 1: Summary of replication workflow cycle time for the period between June 2020 to June 2023.

Cycle Time	S1 ⇨ S2		S2 ⇨ S1	
	# of Cycles	%	# of Cycles	%
<= 1 hour	11,893	73.975	6,605	57.515
<= 2 hours	14,354	89.283	8,925	77.717
<= 4 hours	15,503	96.430	10,435	90.866
<= 8 hours	15,916	98.999	11,152	97.109
<= 12 hours	15,991	99.465	11,327	98.633
<= 24 hours	16,041	99.776	11,434	99.565
> 24 hours	36	0.224	50	0.435

The effect of revisions made to the replication workflow is reflected in Figure 8 and Figure 7, which show the monthly percentage of cycle times completed <=1-hour, <=2-hour, <=4-hour, <=8-hour, <=12-hour, <=24-hour and >24-hour for the period between June 2020 to June 2023. As seen in the graphs, the efficiency of our workflow was low at beginning when RDS was ramping up with increasing number of project directories. This prompted us to change our workflow from *dsync*-ing every project directory in every cycle to *dsync*-ing just the project directories with changes (rev. 2 in Figure 3). We gained some good performance in July 2020 following this change. However, as large number of files were being ingested into those extreme large project directories mentioned in section 3.2, the workflow responded with longer cycle time between August to October 2020 period, before we applied the exception treatments as described in section 3.2 and regained some performance. We saw another performance dip in March 2021 which prompted us to modify the workflow again to *dsync*-ing just the changed subdirectories within project directories (rev. 3 in Figure 3).

There were inevitable situations where the loads were higher on RDS that led to longer cycle time. One major weakness of our workflow is in dealing with replication of renamed files and directories at the source. When a file/directory is renamed, it triggers deletion and recopy of the renamed file/directory from source to destination. For a renamed directory with a lot of files underneath it, that leads to long deletion and recopy of all the files under the renamed directory. This was observed during November 2021 in Figure 8. Nevertheless, 99.688% of the replications were completed within a day.

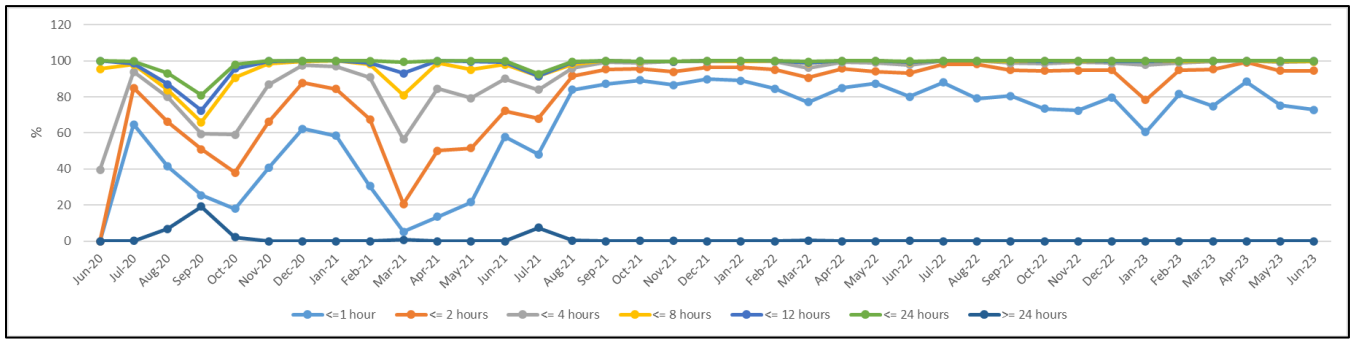

Figure 7: Performance tracking of data replication workflow from S2 to S1 for the period between July 2020 to June 2023.

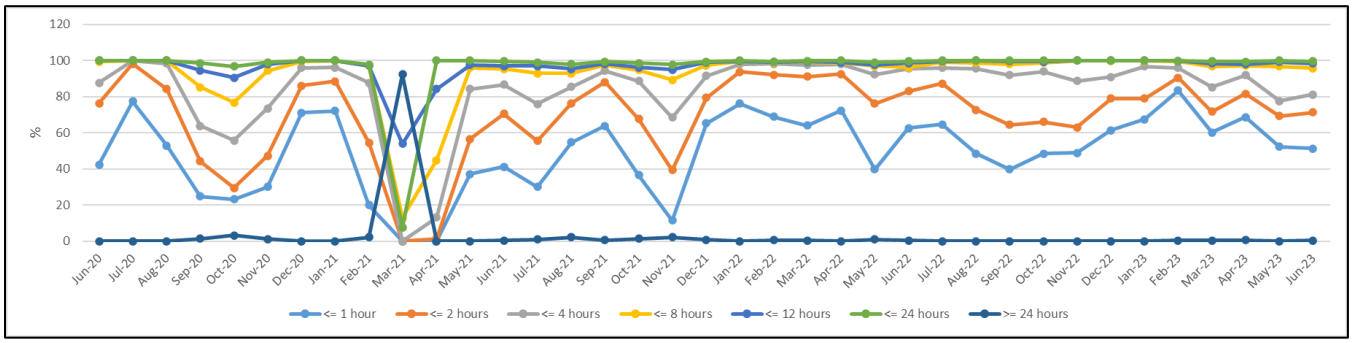

Figure 8: Performance tracking of data replication workflow from S1 to S2 for the period between June 2020 to June 2023.

5. Conclusion

We have been running a home-brewed DR solution at The University of Sydney for the last three years to keep the two 13-petabyte RDS systems in sync using a parallelized data replication workflow. The adaptable and open-source nature of our workflow allows us to finetune it as needed to quickly respond the changing condition of RDS workload. Following a few significant updates to the workflow, we saw improvements to the overall efficiency of the workflow. Nevertheless, there remains areas for improvements. Given the data mover nodes utilized in this replication workflow are the same nodes serving the NFS exports and SMB shares to the end users, we believe the performance of our workflow could be enhanced with greater and dedicated resource availability.

ACKNOWLEDGMENTS

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

REFERENCES

[1] H. Leong, A. Janke, S. Kolmann and D. Richards, "Parallelized Data Replication of Multi-Petabyte Storage Systems," in *SC (SC20) ACM SIGHPC SYSPROS Workshop 2020*, Atlanta, 2023.

[2] A. Tridgell and W. Davison, "The rsync algorithm," The Australian National University, Canberra, 1996.

[3] O. Tange, GNU Parallel 2018, Ole Tange, 2018.

[4] 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.