Integrity Protection for Scientific Workflow Data: Motivation and Initial Experiences

Mats Rynge
Karan Vahi
Ewa Deelman
Information Sciences Institute - University of Southern California

Anirban Mandal
Ilya Baldin
RENCI - University of North Carolina, Chapel Hill

Omkar Bhide
Randy Heiland
Von Welch
Raquel Hill
Indiana University

William L. Poehlman
F. Alex Feltus
Clemson University

ABSTRACT

With the continued rise of scientific computing and the enormous increases in the size of data being processed, scientists must consider whether the processes for transmitting and storing data sufficiently assure the integrity of the scientific data. When integrity is not preserved, computations can fail and result in increased computational cost due to reruns, or worse, results can be corrupted in a manner not apparent to the scientist and produce invalid science results. Technologies such as TCP checksums, encrypted transfers, checksum validation, RAID and erasure coding provide integrity assurances at different levels, but they may not scale to large data sizes and may not cover a workflow from end-to-end, leaving gaps in which data corruption can occur undetected. In this paper we explore an approach of assuring data integrity - considering either malicious or accidental corruption - for workflow executions orchestrated by the Pegasus Workflow Management System. To validate our approach, we introduce Chaos Jungle - a toolkit providing an environment for validating integrity verification mechanisms by allowing researchers to introduce a variety of integrity errors during data transfers and storage. In addition to controlled experiments with Chaos Jungle, we provide analysis of integrity errors that we encountered when running production workflows.

ACM Reference Format:
Mats Rynge, Karan Vahi, Ewa Deelman, Anirban Mandal, Ilya Baldin, Omkar Bhide, Randy Heiland, Von Welch, Raquel Hill, William L. Poehlman, and F. Alex Feltus. 2019. Integrity Protection for Scientific Workflow Data: Motivation and Initial Experiences. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3332186.3332222

PEARC '19, July 28-August 1, 2019, Chicago, IL, USA

ACM ISBN 978-1-4503-7227-5/19/07.
https://doi.org/10.1145/3332186.3332222

1 INTRODUCTION AND MOTIVATION

Researchers are utilizing a variety of computing resources to execute their workflows. It is common to use local resources such as campus cluster, national distributed cyber-infrastructures such as XSEDE [31] or Open Science Grid [27][28], or cloud resources such as Amazon AWS. Executing workflows across different resources usually means adding additional data transfers and storage solutions to the workflow, which can make it more difficult to ensure that data was not corrupted in transit or at rest. At the same time, the operational layers underneath the workflow execution can instill a false sense of security. For example, researchers utilizing workflow technologies might have heard about technologies like TCP checksums, encrypted transfers, checksum validation, RAID and erasure coding, which all imply that the data is well protected from corruption. However, issues such as TCP checksums being too small for modern data sizes [29], possible gaps between technologies and implementations, configuration errors, and software bugs can lead to integrity issues. Here are some examples which are relevant to scientific workflow users:

- A CERN data integrity white paper [24] highlights low level data corruption for data at rest on disk systems to have an error rate at the 10^{-7} level. A similar study by NEC [11] found that 1 in 90 SATA drives will experience silent data corruption.
- Higher level data management tools like Globus Online have explored data corruption scenarios, including the shortcomings of TCP checksumming, and implemented their own user space checksumming [23]
- A bug in Internet2 switches affected scientific data transfers between XSEDE sites [12]: *"XSEDE was notified recently by Internet2 that an error was discovered on the devices that Internet2 uses on its AL2S network that could possibly lead to data corruption. This error could have affected approximately 0.001% of the data that traversed each AL2S device and was undetectable by the standard TCP packet checksum."*
- Another type of data corruption can take place when interactions between different software systems fail. As an example, a Pegasus user reported that her workflow had finished after

spending over 10 years of core hours, with the final output being a zero sized file [19]. The cause of this was a failed transfer with stashcp [33]. Even though the transfer failed, stashcp exited with apparent success - i.e. with an exit code of 0. The faulty exit code signaled success to Pegasus, which in turn decided to clean up all the intermediate workflow data, and thus losing data.

Problems such as these motivated the Scientific Workflow Integrity with Pegasus (SWIP) project [7], funded by the National Science Foundation. A key goal of SWIP is to provide assurance that any changes to input data, executables, and output data associated with a given workflow can be efficiently and automatically detected. Towards this goal, SWIP has integrated data integrity protection into the Pegasus WMS. Integrating these protections into a WMS allows us to take advantage of two aspects of WMSes: (1) they can automatically handle routine tasks such as generating and verifying integrity data which human researchers find tedious and error-prone, and (2) they have a holistic view of a workflow, allowing for integrity verification from end-to-end, catching errors that occur between storage and transfer technologies, and other software systems. We leverage these aspects of WMSes to provide data integrity via the use of cryptographic hashes.

With the above goals in mind we have added new capabilities to Pegasus WMS to automatically generate and track checksums for both when inputs files are introduced and for the files that are generated during execution. Pegasus 4.9.0 was released on October 31st, 2018. This is the first production release with integrity verification, enabled by default (opt-out). Existing Pegasus users who upgrade will automatically receive the benefits from this work.

In addition to integrity protection, the SWIP project has developed Chaos Jungle - a toolkit that provides a controlled environment for integrity experiments by allowing researchers to introduce a variety of predictable network errors. This work is done to validate our integrity protection work, both in terms of functionality and in impact in terms of cost. Functionally, integrity protection can be hard to test since it is easy to mistake a well-functioning system from one that fails to detect errors. Chaos Jungle allows us to predictably introduce errors, ensure that our integrity protection is working, and assess how the protection mechanisms impact performance.

The rest of the paper is organized as follows. In section 2 we detail changes to Pegasus WMS for integrity checking. In sections 3 we give an overview of Chaos Jungle. In section 4, we detail performance overheads of Chaos Jungle evaluated using ExoGENI [15] testbed using Chaos Jungle to inject errors. In section 5 we report back real-world integrity errors detected in production workflows. We end the paper with related work in section 6, future work and and conclusions in section 7 and 8.

2 PEGASUS WORKFLOW MANAGEMENT SYSTEM

Pegasus WMS provides a means for representing the workflow of an application in an abstract form that is independent of the resources available to run it and the location of data and executables. It plans these abstract workflows into an executable form by querying information catalogs. During this planning phase, Pegasus adds data management tasks, which are responsible for staging in input data from user specified locations, moving intermediate data between compute environments, and staging out outputs to a user defined file server. Pegasus also adds data cleanup nodes that remove input and generated output data once the data is no longer needed in the workflow. Pegasus WMS leverages a variety of existing data management tools to do the transfers, dependent on where the data is located, and transfer interfaces available on the compute resources. Tasks on remote compute nodes are managed by lightweight Pegasus instances called PegasusLite. PegasusLite is responsible for staging in the input data for the task to the compute node, launching the task and then staging-out the outputs to the workflow staging site.

The executable workflows are deployed on local or remote distributed resources using the HTCondor DAGMan [30] workflow engine, and often consume tens of thousands CPU hours and involve transfer of terabytes of data. While some data management tools provide checksum-checking capabilities (ensuring that checksum on transferred file is same as that on source), many of the transfer tools do not. Since existing Pegasus users rely on a variety of storage solutions, data access mechanisms, and transfer tools, we have opted to implement end-to-end integrity verification at the Pegasus level whereby the system tracks checksums of every file used and generated as part of the workflow in its information catalog.

2.1 Changes to Pegasus to Implement Integrity Generation and Verification

During the process of planning an abstract workflow to an executable workflow we now add checksum computation and integrity checking steps. We have made extensions to our compute task launching tool *pegasus-kickstart* to generate and publish sha256 checksums for output files created by a task in its provenance record. The generated checksums are populated in Pegasus workflow-monitoring catalog. Users are allowed to specify, along with the file locations, sha256 checksums for raw input data in the replica catalog. A new tool called *pegasus-integrity-check* computes checksums on files and compares them against existing checksum values. PegasusLite has been modified to invoke *pegasus-integrity-check* before launching any computational task. We also have extended our transfer tool *pegasus-transfer* to invoke *pegasus-integrity-check* after completing the transfer of files. Using these new tools and extensions, we have been able to implement integrity checks in the workflows at three levels:

(1) *after the input data has been staged to staging server - pegasus-transfer* verifies integrity of the staged files.
(2) *before a compute task starts on a remote compute node* - This ensures that checksums of the data staged in match the checksums specified in the input replica catalog or the ones computed when that piece of data was generated as part of previous task in the workflow.
(3) *after the workflow output data has been transferred to user servers* - This ensures that output data staged to the final location was not corrupted in transit.

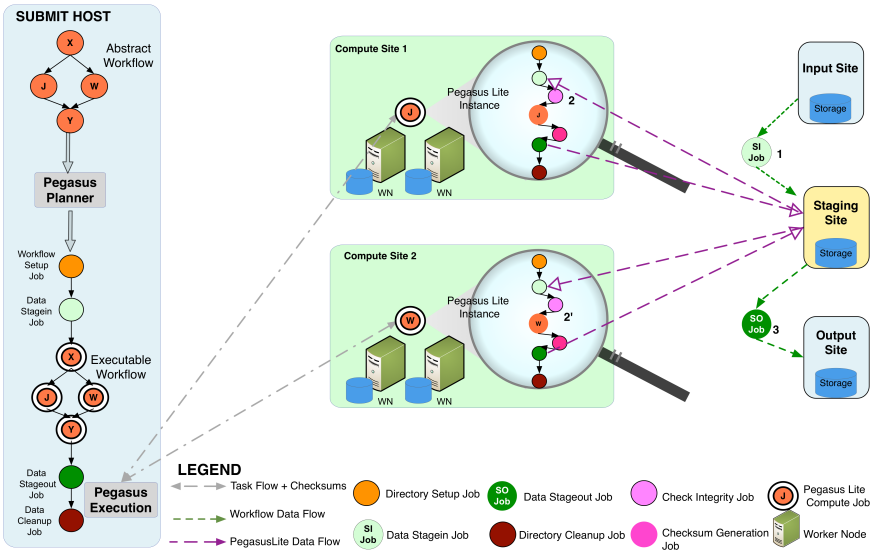

Figure 1: Pegasus integrity verification in non-shared filesystem mode. Note how Pegasus has added integrity verification to the PegasusLite wrapper, after the data transfer to the compute node (2,2'), right before the user specified code is executed. After the code has completed, the produced data is immediately checksummed and the checksum is added to the workflow metadata set, for use in subsequent checksum validations. Checksum validation also happens when input files are staged to the staging server (1) and and after the outputs have been staged to the output site (3)

Figure 1 illustrates the points at which integrity verifications are implemented. In our approach, the reference checksums for the input files for a job are sent to the remote node where a job executes using built-in HTCondor file transfer mechanism. This transfer channel, in theory, may itself suffer from the same limitations of data corruption that is used for data transfers. However, we believe that any corruption in checksum data during transfer will result in a job failing and not erroneously succeeding, which is what we are trying to avoid. For a job to erroneously succeed, both input checksum data has to be corrupted and the actual data file has to be corrupted such that the computed checksum on the remote node for the input file transferred matches the corrupted reference checksum.

2.2 Changes to Capture Overhead of Checksum Calculations

Adding checksum computation and integrity checks to the workflow can result in measurable overheads that affect both the workflow walltime (time to completion of a workflow as seen from workflow submit node) and the total computational CPU units consumed on remote resources. The amount of overhead depends on data set size, workflow structure, data access patterns and the ratio between amount of data and amount of computation.

To better understand these overheads and make the information easily available to our users, we have updated Pegasus monitoring infrastructure to parse this data (the time it takes to compute or verify the checksums) automatically and populate it in the provenance database[32]. The data is captured and aggregated at a job level. For each job, Pegasus records the total time spent on computing checksum or verifying checksum for input and output files, and any failures and retries that happen in the workflow because of caught integrity errors. The data is available via the command line tool *pegasus-statistics*. See section 4.3 for an example. Our motivation to provide this information is two-fold. First, we want to present to our users a clear understanding of the overheads incurred by integrity verifications. Second, we want to communicate to users the benefits incurred by capturing job failures because of corrupted data that may have passed undetected earlier.

3 CHAOS JUNGLE / TESTBED

One of the challenges we faced was how we could reliably and repeatably test integrity protections introduced in Pegasus. For that purpose, we decided to leverage the ExoGENI testbed [15] for creating virtual infrastructure, and then introduce deliberate impairments into the virtual infrastructure to test integrity protection for workflows. ExoGENI allows users to create mutually isolated slices of interconnected infrastructure from multiple independent providers (compute, network, and storage) by taking advantage of existing virtualization mechanisms and integrating various resources together, including Layer-2 dynamic-circuit networks, software defined networks, and private clouds.

The goal of the *Chaos Jungle* software [3] is to introduce different kinds of impairments into the virtual infrastructure - network, compute, storage. In this work, we execute the Chaos Jungle software on virtual infrastructure provisioned by ExoGENI, e.g. on virtual ExoGENI compute nodes used for workflow computations. In this section, we describe how we use Chaos Jungle to introduce a specific kind of impairment - network impairments for data transfers. More specifically, we describe how we mangle data packets during data transfers between virtual compute nodes such that the

checksums still remain valid but data is already corrupted when it reaches the receiver compute node.

We leverage the Linux eBPF (extended Berkeley Packet Filters) functionality [4] for this purpose. eBPF programs can be inserted into various points in a running kernel: *kprobe*, *uprobe*, and various portions of the networking stack (TC [5], XDP [10] etc.). The kernel needs to support this feature and many modern kernels do. The eBPF programs can be used to perform various kinds of profiling unobtrusively e.g. tracing system calls, I/O behavior, network behavior etc. Before being inserted into the kernel, the eBPF programs are checked by a verifier to ensure termination. eBPF programs are written in a language that mimics a subset of C. The BCC toolkit [1] makes it easy to write eBPF programs by combining Python and eBPF programs written in C. The BCC toolchain handles the tasks of compilation and insertion into the kernel.

In our setup, we use either the TC hook or the XDP hook to load an eBPF program into a running kernel at the receiving host. This eBPF program performs the packet mangling. It inspects received packets and modifies some of those that match given flow descriptors described by a common tuple consisting of source, destination IP addreses, protocol number and TCP or UDP port, without affecting the appropriate checksums. The packets thus look valid on the receiving end, however contain invalid data. The difference between attaching to TC hook versus the XDP hook is that while using the TC hook, the eBPF program sees the packets after it enters the networking stack, and hence can leverage the *sk_buff* data structure. While using the XDP hook, the eBPF program sees the packets much earlier - between the network driver and the networking stack, and hence is faster to use; however, it has the disadvantage of a limited NIC driver compatibility, as it requires the driver to pass the received frame in as a single memory page.

4 EVALUATION

In order to validate our implementation for integrity verification in Pegasus we decided on a two-pronged approach. We leveraged the Chaos Jungle to do a series of controlled workflow experiments to demonstrate that as we increase rate of data corruption, the number of job failures that are detected increase. We also deployed a version of Pegasus that had integrity verifications enabled for a subset of our users running large workflows on Open Science Grid, to check if integrity errors are detected in the wild, and determine the overhead on a real science workflow. We first describe the performance overheads of Chaos Jungle in 4.1, before describing the experiments conducted to validate our approach in 4.2. We also evaluate the overhead of checksum calculations in 4.3.

4.1 Performance Overheads of Chaos Jungle

In this section, we present our evaluation of the performance overheads of the Chaos Jungle software itself. We ran the experiments using two VMs on an ExoGENI rack, which were connected by a link of varying bandwidths. The link bandwidth was varied from 100Mb/s to 2000Mb/s. One of the VMs acted as the sender of packets and the other acted as a receiver. The Chaos Jungle software, when used, was executed on the receiver VM to mangle incoming packets. We used the *nuttcp* tool [6] to perform the data transfers, with the *nuttcp* server running on the receiver VM and the *nuttcp* client running on the sender VM. We used the *sar* tool [9] to measure the system CPU utilization on the receiver VM. There were no other activities running on the VMs. We repeated each experiment 10 times. We measured the system CPU utilization during the data transfers for both cases, once with Chaos Jungle enabled and once with Chaos Jungle disabled.

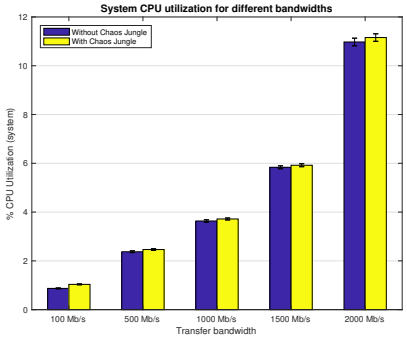

Figure 2: System CPU Utilization overheads of packet mangling with and without Chaos Jungle for different bandwidths.

Figure 2 shows the plots of system CPU utilization for different bandwidths, with and without Chaos Jungle enabled. We observed that the system CPU utilization increases with increasing bandwidth for both cases. This is expected because the kernel networking functions are performing more work as the bandwidth increases. The more important observation is that the overhead of using the Chaos Jungle software is minimal, which is the height difference between the blue and the yellow bars. We observe that the Chaos Jungle overhead is between 1.4 to 3.8% for bandwidths 500 Mb/s and beyond. The overhead is about 19% for 100Mb/s, but with very low absolute values of system CPU utilization, which can be ignored for practical purposes. Hence, we can infer from this result that using the Chaos Jungle software during the experiments described in the following sections, doesn't by itself introduce any significant overheads.

4.2 Chaos Jungle Workflow Experiments

The Chaos Jungle was used to validate and experiment with the Pegasus integrity implementation. The goal was two-fold:

- ensure that Pegasus would detect corrupted data files, and
- handle those failures in a graceful manner.

Chaos Jungle was configured similar to setup described in the previous section, but with one master node, four worker nodes, and a data node that hosted the input data for the workflows. The master and worker nodes constituted the HTCondor pool used by the workflows. Chaos Jungle was executed on the worker nodes. The worker nodes fetched the input data from the data node and were the data receivers in this experiment.

For driving the testing, we used a modified version of a production workflow: OSG-KINC [25] [17] from the Feltus group at Clemson University. OSG-KINC is a systems biology workflow that inputs $N \times M$ floating point gene expression matrices, calculates condition-specific gene correlations for billions of pairwise gene

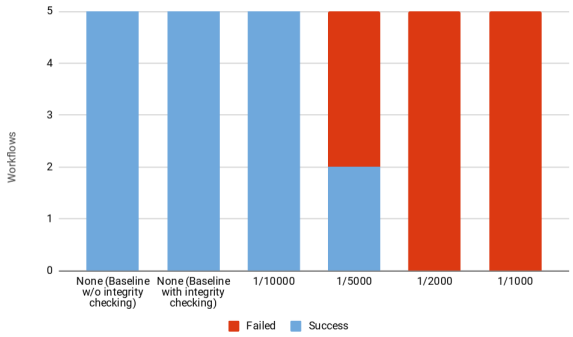

Figure 3: The number of workflows finishing successfully for each Chaos Jungle error rate, and with the number of job tries set to 3.

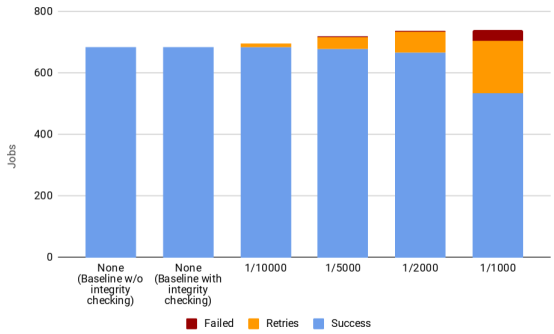

Figure 4: The average number of successful, retried and failed jobs for each Chaos Jungle error rate, and with the number of job tries set to 3. 1/5000 is the breakover case for which 39 of 686 jobs had to be retried and 1 of those exhausted all 3 tries to make the workflow ultimately fail.

combinations, and outputs a gene correlation graph that represents complex interaction patterns of biological molecules. The workflow was configured to use *http* for getting data to the jobs, and *scp* to store data back to the staging site. The choice of protocols here is important, as different protocols have different amounts of built-in protection against low level TCP errors. *http* has no extra built in protection, while *scp* terminates connections with TCP errors due to the end-to-end encryption algorithm. Thus, the errors in this experiment were all due to errors in the data fetching over *http*. Even though we knew the *scp* based transfers would not add to the experiment, the workflow was kept in this configuration as it is the way some users run the workflow in cloud execution environments and we wanted to mimic a real world production setup as far as possible.

We devised six experiments. The first two were baseline runs without any Chaos Jungle error injection, with Pegasus integrity checking enabled and disabled. The remaining four experiments were for the different error injection rates 1/10000, 1/5000, 1/2000 and 1/1000. For example, 1/10000 means that Chaos Jungle would mangle 1 packet out of 10000. For each experiment, 5 workflows were executed, each containing 686 jobs. Pegasus was configured to retry failed jobs a maximum of 3 times.

The purpose of these experiments was to look for error rates for which Pegasus is able to overcome the problem by retrying the failed jobs and as a result executing the workflow to completion successfully, and then error rates for which even retries would not be able to make the workflow complete. Figure 3 and Figure 4 show the results of the experiments. The former shows the 5 workflow executions for each case, and how Pegasus has no problem finishing all 5 runs for the two base line cases and the 1/10000 case. At 1/5000, only 2 out of the 5 workflows manage to succeed, and at 1/2000 and 1/1000 no workflows are able to finish.

Figure 4 highlights averages of the numbers of successful, retried and failed jobs across the 5 runs. The baseline cases only contain successful jobs. The 1/10000 case finished successfully, after an average of 12 job retries. For 1/5000, which is the breakover case of 2 out of 5 workflows finishing, we have an average of 39 job retries, and 1 job failing 3 times. As expected, the 1/2000 and 1/1000 cases have even higher job retries (67 and 172) and failed jobs (4 and 34).

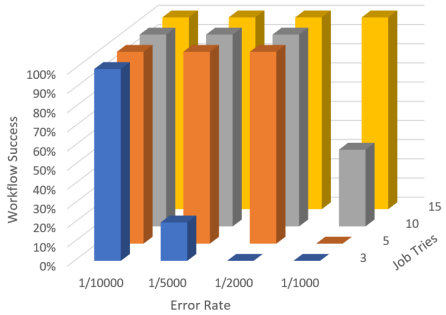

Figure 5: By increasing the number of allowed job tries, the workflows can be made to finish successfully even under the higher error injection rates.

With these results in mind, we used Chaos Jungle to explore another dimension of the tests. By increasing the number of allowed job tries, workflows could be made successful even under the higher error injection rates. These results can be seen in Figure 5. Note that for production workflows, we do not recommend high job retries settings, as there are many reason a job could be retried, and integrity errors are just one of the reasons. In most cases, it is better for the workflow to stop execution after a small number of retries, and have the user examine the problem. This is easily done with the included *pegasus-analyzer* tool, and if it is determined that the workflow should be given more retries, the workflow can be restarted from the current state but with a new set of retries with the *pegasus-run* command.

4.3 Overhead of Checksum Calculations

A concern when augmenting a workflow with additional steps for calculating and verifying the checksums is the introduced execution time overhead. For synthetic workflows, we can see an overhead of 7%. However, in production workflows which has a much higher compute time to data size ratio, we have found the overhead to be much lower than 1%. To illustrate this point, consider the results

from a 1,000 job OSG-KINC production workflow. The results from *pegasus-statisitcs* on this run can be found in Listing 1.

For this workflow, 14 minutes of computing time was used computing and comparing checksums for the files referred to in the workflow in contrast to 17 days and 23 hours of computing time to run the workflow. This results in an overhead of 0.054% incurred by the integrity verification.

A second example is Ariella Gladstein's population modeling workflow [20]. A 5,000 job workflow used up 167 days and 16 hours of core hours, while spending 2 hours and 42 minutes doing checksum verification, with an overhead of 0.068%.

A smaller example is the Dark Energy Survey Weak Lensing Pipeline [16] with 131 jobs. It used up 2 hours and 19 minutes of cumulative core hours, and 8 minutes and 43 seconds of checksum verification. The overhead was 0.062%.

While we believe the examples above are representative of average compute intensive workflows, the overhead will vary from user to user due to factors like workflow structure, and the ratio of compute time to data size.

5 INTEGRITY ISSUES DETECTED IN PRODUCTION WORKFLOWS

When the SWIP project was initiated, an open question was how frequent real-world integrity errors would be detected. One extreme outcome could be that the integrity issues would always be caught by the infrastructure, and Pegasus would never be the one detecting the issues. Due to this concern, we selected a set of friendly users to start executing with the new version of Pegasus. The users were selected based on their workload (large number of jobs in a distributed execution environment, resulting in a large number of data transfers), as well as the data transfer protocols used (no encryption to aid the infrastructure detecting integrity issues, plain http for example). Using these early adopters, we have been able to detect a set of real-world integrity issues.

One user affected is the Feltus group at Clemson University, running the OSG-KINC [25] and OSG-GEM [26] workflows on Open Science Grid. OSG-GEM calculates the floating point matrix input for OSG-KINC from multi-GB DNA deep sequence datasets. We have found two production instances where Pegasus detected integrity problems.

The first instance, *kinc-1522378583*, was executed in April 2018. It had 50,606 jobs, ran for 3 days, and used up almost 4 years of cumulative CPU core hours. The workflow finished successfully after 224 automatic job retries. Of those job retries, 60 were from detected integrity checksum errors. The 60 integrity errors all happened across 3 compute nodes:

- 1 input file error at University of Colorado.
- 3 input file (kinc executable) errors at University of Nebraska. The timespan across the failures was 16 seconds.
- 56 input file errors on a different compute node at University of Nebraska. The timespan across the failures was 1,752 seconds.

The grouping of the errors, both spatial and temporal, can be explained by the data transfer tools used in this case, stashcp and CVMFS [33]. Data is cached at the node level, compute site level, and in regional caches. As CVMFS does checksumming during the data transfers, we suspect that cache corruption is to blame, at least for case 2 and 3. That would explain why the same file kept failing on the same node for a short period of time. However, definite proof of this theory was impossible to obtain as the problematic files had been purged from the cache when the error analysis took place.

A second production workflow failed in May of 2018. This time it was OSG-GEM (instance *osg-gem-1525147773*), an even larger workflow with 168,678 jobs and over 3.2 million data transfers. Pegasus detected 331 integrity errors and, just like the previous workflow, the errors are grouped in a manner which implies cache corruption. 171 errors were on the same site, same file, with the same corrupt checksum (that is, the same broken file was downloaded over and over again), over a time span of 5 hours and 20 minutes. The remaining 160 errors are slightly more spread out, but still has some patterns to them. For example, 93 of those were at at University of Connecticut. In that set, the files and checksum differed, but being clustered like that at one site points to a local infrastructure problem.

It is important to point out that stashcp and using CVMFS for data distribution like this is something still under active development on the Open Science Grid. We have brought these findings to the developers. It is unclear if we have hit the same bug over and over again in the system, or if the errors shows a larger infrastructure problem in the distribution and caching of the data. The take away message is that by turning on integrity verification, Pegasus protected the researcher's data by detecting the errors, enabling the workflow to continue using job retries, and providing a report of the issues encountered on a run which, from the point of view of the researcher, was a successful workflow execution.

Another user, Nepomuk Otte from the VERITAS project [13], is using the Open Science Grid to compare the recorded images from the VERITAS telescope with simulated ones to find out if a shower was produced by an actual gamma ray or a cosmic ray, which would be a background event. One difference compared to the Feltus' workflows, is that intermediate files in the workflow are stored in a CEPH object store, and accessed via the provided S3-compatible interface [2]. Similar to the stashcp transfer, S3 in this setup is using plain http, but there is no caching of the data. Therefore, we do not see the same grouping and the related clues when integrity errors do happen. The errors are more evenly distributed. For example, one workflow consisting of 120,000 jobs, detected 48 integrity errors. This comes out to a 0.04% error rate, or about 1 in 2,500 transfers failing. Due to the size of these workflows, it takes time until enough have been run to gather statistically significant numbers. Thus, we consider these interesting anecdotal data points, but there is not yet enough data to draw specific conclusions about infrastructure wide failure rates. Regarding the cause of the issues, there are a set of possibilities such as corruption introduced in the plain http stream or bugs in the CEPH S3 gateway or the *pegasus-s3* command line tool. Determining, diagnosing and reporting the causes to infrastructure operators or developers will be addressed in a project follow-on, but in the meantime, note that just like the previous example, the 48 errors were properly detected and handled with job retries, resulting in a successful workflow execution.

```
Type           Succeeded  Failed  Incomplete  Total     Retries   Total+Retries
Jobs           1606       0       0           1606      31        1637

Workflow wall time                                       : 7 hrs, 59 mins
Cumulative job wall time                                 : 17 days, 23 hrs
# Integrity Metrics
3944 files checksums compared with total duration of 9 mins, 18 secs
1947 files checksums generated with total duration of 4 mins, 37 secs
# Integrity Errors
Failures: 0 jobs encountered integrity errors
```

Listing 1: Output of the pegasus-statistics tool showing the checksum overhead

6 RELATED WORK

This project touches on several broad topics: data integrity, workflows, testing environments, software security, etc., each of which has a significant amount of related work. For example, secure hashing algorithms, used to ensure data integrity, have evolved for over 30 years. However, for the specific combination of topics that we address, there seems to be little related work. In a prototype project involving the Kepler scientific workflow engine [21], a data integrity check was added consisted of "on-hash" and "post-hash" operations; the former when data is being used, the latter when the workflow terminates. In related work, the authors assessed the runtime overhead of their approach [22]. However, to our knowledge, the integrity checking was never incorporated into Kepler itself.

Data integrity for data at rest has seen plenty of research and implementations, such as for ZFS [34] and cloud object stores [14]. Integrity of data in transit has been explored for example by the Globus Online team [18] as well as integrated into the secure file copy (scp) tool. What we are arguing in this paper is that having trustworthy storage and transfers are just pieces of the puzzle - what computational scientists need are end-to-end solutions during the full lifetime of the scientific data.

Large scientific collaborations such as LIGO Scientific Collaboration recognize the importance of data integrity and have taken steps to build checksumming[8] into their frame file format that is used to describe the data coming from the LIGO detectors. LIGO analysis code checks for checksum when working with these frame files. Checksums are computed and similarly enforced only for some (but not all) intermediate and derived data products generated by the analysis pipelines, but not for the executables or workflow metadata.

7 FUTURE WORK

In this paper we have shown that data integrity can be a major issue for computational based research. We have implemented an end-to-end solution in the Pegasus WMS, provided experimental data from our testbed, and analysis of data integrity problems of real world production workflows. Contemplating these issues, implementing, and testing our approach has been a wonderful mix of computer science and software engineering. We are aware that there is still much work to be done.

Our work so far has raised some interesting questions:

What is the impact of these integrity errors? It is an interesting intellectual exercise to contemplate what the impact of these errors could have been on the workflow if they were not detected by Pegasus. There is no guarantee that the code would even have detected the issue. The error detected when downloading the executable could have been in, for example, a conditional section or function which would not have been executed with the given options and data. Or, corruption could have lead to a fatal exception. Worst case would have been a silent error which affected the computation. Similarly, input data corruption could have been detected by the code, not had any impact, or introduced faulty data which could have produced faulty results silently.

How should responsibility for integrity be distributed? Ultimately, the goal is to provide data integrity protection for the entirety of a research workflow. While a WMS can provide such protections for the portions of the workflow it manages, many scientific workflows will have aspects that are beyond the control of the WMS. This implies the responsibility must be shared between the researcher and the CI provider. If so, the delineation for this responsibility is not clear at this time, meaning there are potential gaps in workflows where responsibility for data integrity is not clear. In so far as data integrity is a necessary aspect of reproducibility, this implies a risk to reproducibility.

How does one diagnosis an integrity error? An open question we are contemplating is how much effort to put into diagnosing the source of integrity errors. Our goal is to provide integrity of workflow data, but as a side effect we are detecting what may be serious issues in the underlying infrastructure which ideally would be reported to an appropriate operator to be addressed. However our real world experience has demonstrated, as described in section 5, more data is needed to diagnose the source of a problem as opposed to simply retrying when a problem is detected. Hence, we plan to develop an integrity analysis framework that collects integrity relevant data from the infrastructure and applications, and utilizes offline and online ML-based algorithms to automatically detect, analyze and pinpoint source of integrity anomalies. To develop the framework, we plan to engage in an integrated approach of testbed experimentation, ML model training using workflow data on testbeds, and ML model validation on production CI for real world workflows. We plan to integrate more features into Chaos Jungle to enable experimentation with data corruption at multiple layers and within multiple subsystems. As our investigation demonstrated, it is important to test and validate all elements of the infrastructure for resilience to data corruption, thus requiring a complex environment that allows us to perform repeatable experiments with different combinations of conditions.

At this point, Pegasus can do integrity verification for workflows executed in non-shared filesystem deployments, where jobs are launched using the lightweight job wrapper Pegasus Lite on the remote worker node. However, Pegasus Lite is not used to launch jobs in certain shared filesystem deployments such as when launching MPI jobs. For such scenarios, we plan to update Pegasus to add separate data integrity check jobs to the workflow. This will allow compute jobs to start in parallel with data integrity jobs, and a workflow failure will be triggered if any of the data integrity jobs fail.

8 CONCLUSION

Our work has added integrity protection to the Pegasus WMS and demonstrated data integrity errors occurring on cyberinfrastructure. This raises a question about the extent to which cyberinfrastucture can guarantee data integrity and, if it cannot, what the impact of data integrity flaws will be and how responsibility for data integrity should be distributed between researchers and cyberinfrastructure operators.

ACKNOWLEDGMENTS

The Scientific Workflow Integrity with Pegasus (SWIP) Project is supported by the National Science Foundation under grant 1642070, 1642053, and 1642090. Pegasus is funded by The National Science Foundation under OAC SI2-SSI program, grant 1664162. This research was done using resources provided by the Open Science Grid [27] [28], which is supported by the National Science Foundation award 1148698, and the U.S. Department of Energy's Office of Science. The views expressed do not necessarily reflect the views of the National Science Foundation or any other organization.

REFERENCES

[1] [n. d.]. BPF Compiler Collection (BCC). https://github.com/iovisor/bcc.
[2] [n. d.]. CEPH Object Gateway. http://docs.ceph.com/docs/mimic/radosgw/.
[3] [n. d.]. Chaos Jungle. https://github.com/RENCI-NRIG/chaos-jungle.
[4] [n. d.]. Linux Enhanced BPF (eBPF) Tracing Tools. http://www.brendangregg.com/ebpf.html.
[5] [n. d.]. Linux Traffic Control. https://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/.
[6] [n. d.]. nuttcp. https://www.nuttcp.net/.
[7] [n. d.]. Scientific Workflow Integrity with Pegasus project. https://cacr.iu.edu/projects/swip/.
[8] [n. d.]. Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detectors (IGWD). https://dcc.ligo.org/LIGO-T970130/public/main.
[9] [n. d.]. SYSSTAT Utilities. http://sebastien.godard.pagesperso-orange.fr/documentation.html.
[10] [n. d.]. XDP - eXpress Data Path. https://prototype-kernel.readthedocs.io/en/latest/networking/XDP/.
[11] 2009. *Silent data corruption in disk arrays: A solution.* Technical Report. https://www.necam.com/docs/?id=54157ff5-5de8-4966-a99d-341cf2cb27d3
[12] 2012. *XSEDE Network Status.* Technical Report. https://portal.xsede.org/user-news/-/news/item/6390
[13] V. A. Acciari, M. Beilicke, G. Blaylock, S. M. Bradbury, J. H. Buckley, V. Bugaev, Y. Butt, O. Celik, A. Cesarini, L. Ciupik, P. Cogan, P. Colin, W. Cui, M. K. Daniel, C. Duke, T. Ergin, A. D. Falcone, S. J. Fegan, J. P. Finley, G. Finnegan, P. Fortin, L. F. Fortson, K. Gibbs, G. H. Gillanders, J. Grube, R. Guenette, G. Gyuk, D. Hanna, E. Hays, J. Holder, D. Horan, S. B. Hughes, M. C. Hui, T. B. Humensky, A. Imran, P. Kaaret, M. Kertzman, D. B. Kieda, J. Kildea, A. Konopelko, H. Krawczynski, F. Krennrich, M. J. Lang, S. LeBohec, K. Lee, G. Maier, A. McCann, M. McCutcheon, J. Millis, P. Moriarty, R. Mukherjee, T. Nagai, R. A. Ong, D. Pandel, J. S. Perkins, M. Pohl, J. Quinn, K. Ragan, P. T. Reynolds, H. J. Rose, M. Schroedter, G. H. Sembroski, A. W. Smith, D. Steele, S. P. Swordy, A. Syson, J. A. Toner, L. Valcarcel, V. V. Vassiliev, S. P. Wakely, J. E. Ward, T. C. Weekes, A. Weinstein, R. J. White, D. A. Williams, S. A. Wissel, M. D. Wood, and B. Zitzer. 2008. Observation of Gamma-Ray Emission from the Galaxy M87 above 250 GeV with VERITAS. *The Astrophysical Journal* 679, 1 (2008), 397. https://doi.org/10.1086/587458
[14] M. F. Al-Jaberi and A. Zainal. 2014. Data integrity and privacy model in cloud computing. In *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*. 280–284. https://doi.org/10.1109/ISBAST.2014.7013135
[15] Ilya Baldin, Jeff Chase, Yufeng Xin, Anirban Mandal, Paul Ruth, Claris Castillo, Victor Orlikowski, Chris Heermann, and Jonathan Mills. 2016. ExoGENI: A multi-domain infrastructure-as-a-service testbed. In *The GENI Book*. Springer, 279–315.
[16] Chihway Chang, Michael Wang, and Scott et al Dodelson. 2019. A unified analysis of four cosmic shear surveys. *Monthly Notices of the Royal Astronomical Society* 482, 3 (2019), 3696–3717. https://doi.org/10.1093/mnras/sty2902
[17] Stephen P. Ficklin, Leland J. Dunwoodie, William L. Poehlman, Christopher Watson, Kimberly E. Roche, and F. Alex Feltus. 2017. Discovering Condition-Specific Gene Co-Expression Patterns Using Gaussian Mixture Models: A Cancer Case Study. *Scientific Reports* 7, 1 (2017), 8617. https://doi.org/10.1038/s41598-017-09094-4
[18] I. Foster. 2011. Globus Online: Accelerating and Democratizing Science through Cloud-Based Services. *IEEE Internet Computing* 15, 3 (May 2011), 70–73. https://doi.org/10.1109/MIC.2011.64
[19] Ariella Gladstein and Mats Rynge. 2017. Personal email communication.
[20] Ariella L Gladstein and Michael F Hammer. 2018. Substructured population growth in the Ashkenazi Jews inferred with Approximate Bayesian Computation. (2018). https://doi.org/10.1093/molbev/msz047
[21] D. Kim and M. A. Vouk. 2015. Securing Scientific Workflows. In *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*. 95–104. https://doi.org/10.1109/QRS-C.2015.25
[22] Donghoon Kim and Mladen A Vouk. 2016. Assessing Run-time Overhead of Securing Kepler. *Procedia Computer Science* 80, C (2016), 2281–2286.
[23] S. Liu, E. S. Jung, R. Kettimuthu, X. H. Sun, and M. Papka. 2016. Towards optimizing large-scale data transfers with end-to-end integrity verification. In *2016 IEEE International Conference on Big Data (Big Data)*. 3002–3007. https://doi.org/10.1109/BigData.2016.7840953
[24] Bernd Panzer-Steindel. 2007. *Data Integrity.* Technical Report. https://indico.cern.ch/event/13797/contributions/1362288/attachments/115080/163419/Data_integrity_v3.pdf
[25] W. L. Poehlman, M. Rynge, D. Balamurugan, N. Mills, and F. A. Feltus. 2017. OSG-KINC: High-throughput gene co-expression network construction using the open science grid. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 1827–1831. https://doi.org/10.1109/BIBM.2017.8217938
[26] William L. Poehlman, Mats Rynge, Chris Branton, D. Balamurugan, and Frank A. Feltus. 2016. OSG-GEM: Gene Expression Matrix Construction Using the Open Science Grid. *Bioinformatics and Biology Insights* 10 (2016), BBI.S38193. https://doi.org/10.4137/BBI.S38193
[27] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank WÃijrthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick. 2007. The Open Science Grid. *Journal of Physics: Conference Series* 78, 1 (2007), 012057. https://doi.org/10.1088/1742-6596/78/1/012057
[28] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein. 2009. The Pilot Way to Grid Resources Using glideinWMS. In *2009 WRI World Congress on Computer Science and Information Engineering*, Vol. 2. 428–432. https://doi.org/10.1109/CSIE.2009.950
[29] J. Stone, M. Greenwald, C. Partridge, and J. Hughes. 1998. Performance of checksums and CRCs over real data. *IEEE/ACM Transactions on Networking* 6, 5 (Oct 1998), 529–543. https://doi.org/10.1109/90.731187
[30] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience* 17, 2-4 (2005), 323–356.
[31] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 5 (Sept.-Oct. 2014), 62–74. https://doi.org/10.1109/MCSE.2014.80
[32] K. Vahi, I. Harvey, T. Samak, D. Gunter, K. Evans, D. Rogers, I. Taylor, M. Goode, F. Silva, E. Al-Shkarchi, G. Mehta, A. Jones, and E. Deelman. 2012. A General Approach to Real-Time Workflow Monitoring. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 108–118. https://doi.org/10.1109/SC.Companion.2012.26
[33] Derek Weitzel, Brian Bockelman, Dave Dykstra, Jakob Blomer, and Ren Meusel. 2017. Accessing Data Federations with CVMFS. *Journal of Physics: Conference Series* 898, 6 (2017), 062044. https://doi.org/10.1088/1742-6596/898/6/062044
[34] Yupu Zhang, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2010. End-to-end Data Integrity for File Systems: A ZFS Case Study. In *FAST*.