Integrity Protection for Scientific Workflow Data: Motivation and Initial Experiences

Mats Rynge, Karan Vahi, Ewa Deelman
Anirban Mandal, Ilya Baldin
Omkar Bhide, Randy Heiland, Von Welch, Raquel Hill
William L. Poehlman, F. Alex Feltus

Information Sciences Institute - University of Southern California
RENCI - University of North Carolina, Chapel Hill
Indiana University
Clemson University

The SWIP Project is supported by the National Science Foundation under Grant 1642070, 1642053, and 1642090.
Pegasus Workflow Management System, Production Use

Last 12 months: Pegasus users ran 240K workflows, 145M jobs

Majority of these include data transfers, using LAN, the Internet, local and remote storage

https://pegasus.isi.edu/
Goals:

Provide additional assurances that a scientific workflow is not accidentally or maliciously tampered with during its execution.

Allow for detection of modification to its data or executables at later dates to facilitate reproducibility.

Integrate cryptographic support for data integrity into the Pegasus Workflow Management System.

PIs: Von Welch, Ilya Baldin, Ewa Deelman, Raquel Hill
Team: Omkar Bhide, Rafael Ferreira da Silva, Randy Heiland, Anirban Mandal, Rajiv Mayani, Mats Rynge, Karan Vahi
Our Talk

- Introduction and Motivations
- Our Approach
- Current Status
- Welcome to the Jungle
- Integrity Issues in the Wild
- Future Work
Data Integrity

010110101010

010100101010

010100101010
Challenges to Scientific Data Integrity

Modern IT systems are not perfect - errors creep in.

At modern “Big Data” sizes we are starting to see checksums breaking down.

Plus there is the threat of intentional changes: malicious attackers, insider threats, etc.

User Perception: “Am I not already protected? I have heard about TCP checksums, encrypted transfers, checksum validation, RAID and erasure coding – is that not enough?”
Motivation:
CERN/NEC Studies of Disk Errors

Examined Disk, Memory, RAID 5 errors.

“The error rates are at the 10^{-7} level, but with complicated patterns.” E.g. 80% of disk errors were 64k regions of corruption.

Explored many fixes and their often significant performance trade-offs.

A similar study by NEC found that 1 in 90 SATA drives will experience silent data corruption.

https://www.necam.com/docs/?id=54157f5-5de8-4966-a99d-341cf2cb27d3
Motivation: Network Corruption

Network router software inadvertently corrupts TCP data and/or checksum!

XSEDE and Internet2 example from 2013.

Second similar case in 2017: University of Chicago network upgrade caused data corruption for the FreeSurfer/Fsurf project.

https://www.xsede.org/news/-/news/item/6390
Motivation:
Software failures

Bug in StashCache data transfer software would occasionally cause silent failure (failed but returned zero).

Failures in the final staging out of data were not detected.

The workflow management system, believing workflow was complete, cleaned up. With the final data being incomplete and all intermediary data lost, ten CPU-years of computing came to naught.

How is this an data integrity issue? The workflow system should have verified that the data at the storage system after the transfer, is the expected data.
Our Talk

- Introduction and Motivations
- **Our Approach**
- Current Status
- Welcome to the Jungle
- Integrity Issues in the Wild
- Future Work
Our High Level Plan...

- Workflow Management Systems (WMS) are great places to tackle data integrity.
- They understand what data is created and ingested and do not mind tedious tasks such as generating and checking checksums.
- Placement is important within the workflow of generate/validate checksums.
- Pegasus WMS is widely used (LIGO, SCEC, SoyKB, Montage, etc.) by the scientific community and is the target of our improvements.
Application-level Checksums – SHA256

• Application-level checksums (hashes) allow for detection of changes.

• Explored some more advanced solutions, but at the end simplicity won

• Checksums already in use by many data transfer applications: scp, Globus/GridFTP, some parts of HTCondor, etc, but SWIP is focusing on end-to-end as well as over longer time periods

e.g. using a SHA in Python:

```python
>>> hashlib.sha256(b"The Answer to the Ultimate Question of Life, the Universe, and Everything is 42").hexdigest()
'8a72856cf94464dd641f0a2620ab604dd7a3f50293784a3a399acf6dc5b651cb'
```

```python
>>> hashlib.sha256(b"The Answer To the Ultimate Question of Life, the Universe, and Everything is 42").hexdigest()
'a39be9fd272f2569aa95a07134a55f032ecb5c51cef6d66fe4032ec30bf4f1b6'
```

```python
>>> hashlib.sha256(b"The Answer is 42").hexdigest()
'cbf296e175f02156cd60d6bf93aebd92893e72a0c4c48eade092d0dc7e28fc1'
```
Our Talk

- Introduction and Motivations
- Our Approach
- **Current Status**
- Welcome to the Jungle
- Integrity Issues in the Wild
- Future Work
Integrity validation is on by default since the Pegasus 4.9.0 release (Oct 31\textsuperscript{st}, 2018). Users who upgrade will automatically get the protection, but can opt out.

Sharing of detailed monitoring data with the Pegasus team is off by default. Users can opt-in. (We will come back to this at the end of the talk)
Pegasus performs integrity checksums on input files right before a job starts on the remote node.

- For raw inputs, checksums specified in the input replica catalog along with file locations
- All intermediate and output files checksums are generated and tracked within the system.
- Support for sha256 checksums

Job failure is triggered if checksums fail
Our Talk

- Introduction and Motivations
- Our Approach
- Current Status
- Welcome to the Jungle
- Integrity Issues in the Wild
- Future Work
How do you know your integrity protection is working?

• Imagine the following: You finish adding integrity protection to your software. You run a workflow and all goes smoothly.

• Was there no integrity problem or did you just fail to detect it?

• How do you reliably and repeatedly test integrity protection?
Confidence in the implementation: Bamboo

- At commit, for each target platform:
  1. Build binary, workers, RPMs, DEBs, ....
  2. Run unit tests for Java, Python, and C components
  3. ~ 100 unit tests
- Nightly:
  1. Run functional tests. These are full workflows, configured to provide good code coverage
  2. ~ 85 workflows
Enter the Chaos Jungle!

https://github.com/RENCI-NRIG/chaos-jungle

Inspired by Netflix’s Chaos Monkey.
https://github.com/Netflix/chaosmonkey

Goal of Chaos Jungle (CJ) is to introduce different kinds of impairments into the virtual infrastructure - network, compute, storage.

The RENCI ORCA software creates virtual infrastructure on ExoGENI testbed. CJ software introduces impairments into data transfers.

We get virtual infrastructure that intentionally corrupts data

Randomly or predictably?

Now we can test how software runs under bad conditions.
Chaos Jungle

Uses Linux eBPF (extended Berkeley Packet Filters) functionality

Introduces a small eBPF program into the kernel attaching to either TC filter or XDP hooks

Inspects received packets and modifies some of those that match flow descriptors without affecting the appropriate checksums.

The packets thus look valid on the receiving end, however contain invalid data.

Fast and performant.

https://github.com/RENCI-NRIG/chaos-jungle
Chaos Jungle Experiment Setup

1. Launch workflow with Pegasus integrity checking enabled

2. Workflow data is fetched from http server hosted on Data node

3. Integrity check errors appear as events in the Grafana dashboard

0. Chaos jungle scripts executed on the HTCondor workers; Script mangles packets while preserving checksum
Total Events: 722
Job Failures: 20
Integrity Failures: 10
Cumulative Wait Time: 20 minutes
Our Talk

- Introduction and Motivations
- Our Approach
- Current Status
- Welcome to the Jungle
- **Integrity Issues in the Wild**
- Future Work
Production Workflows

• Large workflows with lots of data transfers

• “Unprotected” protocols - no SSL or other protocol level protections

• Open Science Grid - WAN transfers

• Collecting the data is on an opt-in basis
Initial Results with Integrity Checking on

• OSG-KINC workflow (50,606 jobs) encountered 60 integrity errors in the wild (production OSG). The problematic jobs were automatically retried and the workflow finished successfully.

• The 60 errors took place on 3 different hosts. The first one at UColorado, and group 2 and 3 at UNL hosts.

• Error Analysis (by hand)
  • 1 input file error at University of Colorado.
  • 3 input file (kinc executable) errors on one node at University of Nebraska. The timespan across the failures was 16 seconds. We suspect that the node level cache got corrupted.
  • 56 input file errors on a different compute nodes at University of Nebraska. The timespan across the failures was 1,752 seconds. We suspect that the site level cache got corrupted.
Initial Results – VERITAS / Nepomuk Otte, GATech

Seeing very small, but steady stream of corrected integrity errors from reporting back to Pegasus dashboard.

For VERITAS, ~.04% of transfers fail with integrity errors. (~1 / 2500 transfers)

Cause uncertain (diagnosis is harder than detection).

Possibly errors in http based transfers (s3 protocol against CEPH)
Checksum Overheads

• We have instrumented overheads and are available to end users via pegasus-statistics.

<table>
<thead>
<tr>
<th>Type</th>
<th>Succeeded</th>
<th>Failed</th>
<th>Incomplete</th>
<th>Total</th>
<th>Retries</th>
<th>Total+Retries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs</td>
<td>1606</td>
<td>0</td>
<td>0</td>
<td>1606</td>
<td>31</td>
<td>1637</td>
</tr>
</tbody>
</table>

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

• Other sample overheads on real world workflows
  • Ariella Gladstein’s population modeling workflow
    • 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 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%.
Our Talk

- Introduction and Motivations
- Our Approach
- Current Status
- Welcome to the Jungle
- Integrity Issues in the Wild
- Future Work
Challenges

• Can we do more than know “something changed?”

• Detecting error easier than diagnosing error.

• Balance performance / integrity trade-off?

• How do we handle storage without compute capabilities?

• Long data life: today’s cryptographic algorithms will probably not last as long as we need the science data. E.g. what threats will Quantum computing bring?

• When do we hit limits of cryptographic algorithms (collisions)?

• Are all errors in all types of data of equal concern?
Going Forward: Integrity Introspection for Scientific Workflows (IRIS)

• National Science Foundation CICI IRIS Grant #1839900

• SWIP addresses *integrity checking* making sure that workflow computations are protected from integrity errors, but
  — Doesn't address analysis of integrity errors discovered, i.e. tracing the source of error or doing root cause analysis to remedy the problem.

• IRIS goal: Detect, diagnose, and pinpoint the source of unintentional integrity anomalies in scientific workflow executions on distributed cyberinfrastructure. *(integrity analysis)*
IRIS Overall Approach

IRIS Overview

Train ML algorithms on controlled testbeds and validate on national CI by integrating framework with Pegasus.

Engage with science application partners to deploy the analysis framework.

IRIS proposed framework
Thanks!

We thank the National Science Foundation for funding this work (Grants 1642070, 1642053, 1642090). Views expressed may not necessarily be the views of the NSF. Thanks to Eli Dart for Brocade TSB details.