TRUSTED CI
THE NSF CYBERSECURITY
CENTER OF EXCELLENCE

An Examination and Survey of Random Bit Flips and Scientific Computing

Originally published December 20, 2019
Revised November 11, 2020
v0.13 — Revised Final Report
For Public Distribution

Reinhard Gentz and Sean Peisert

About Trusted CI

The mission of Trusted CI is to provide the NSF community with a coherent understanding of cybersecurity, its importance to computational science, and what is needed to achieve and maintain an appropriate cybersecurity program.

Acknowledgments

Our sincere thanks to Andrew Adams, Jim Basney, Robert Cowles, Mark Krenz, and Von Welch for their extensive feedback on this document and the ideas presented in it.

This document is a product of the Center for Trustworthy Scientific Cyberinfrastructure (Trusted CI). Trusted CI is supported by the National Science Foundation under Grants 1234408, 1547272 and 1920430. For more information about the Center for Trustworthy Scientific Cyberinfrastructure please visit: http://trustedci.org/.

Using & Citing this Work

Cite this work using the following information:
Reinhard Gentz and Sean Peisert, "An Examination and Survey of Random Bit Flips and Scientific Computing," Trusted CI Technical Report, originally published December 20, 2019, revised November 11, 2020.

http://hdl.handle.net/2022/24910

This work is available on the web at the following URL:
http://trustedci.org/reports

Table of Contents

Executive Summary

Data error, modification, and loss can occur in scientific computing due to numerous potential causes, including those natural, accidental, or malicious in nature. Such errors and modifications can silently impact scientific computing results if not detected and corrected or compensated for through other means. The goal of this document is to describe the ways in which integrity faults due to bit flips occur, their potential odds of occurring, and potential mitigation mechanisms in high-level, practical terms, and is broken into individual computer components and environments describing each of those elements. We conclude the report by summarizing key issues and several best practices for mitigation of the effects of bit flip-induced errors.

1 Background

Data error, modification, and loss can occur in scientific computing for numerous reasons, including, but not limited to improper scientific design, mis-calibrated sensors, and outright bugs including unaccounted-for non-determinism in computational workflows, improper roundoff and truncation errors, hardware failures, and "natural" faults. In addition, numerous other sources of "error" beyond "bit flips" can arise with regard to reproducibility, including inability to recreate the same software and/or hardware conditions that gave the original results, such as legacy code incompatible with modern systems, among other conditions.

Nonetheless, "bit errors" in data files can and do happen, and can be classified in three types types: (1) isolated single errors due to some kind of interference causing bit flips; (2) bursty faults of a number of sequential bits, due to some kind of mechanical failure or electrical interference; (3) malicious tampering. This document focuses on the first, thereby excluding both mechanical failures and malicious tampering, including malicious tampering that causes bit flips, such as Rowhammer type attacks [KDK+14]. The goal of this document is to describe the ways integrity faults occur, their potential odds of occurring, and potential mitigation mechanisms in high-level, practical terms, and is broken into individual computer components and environments describing each of those elements.

We note that this document is not a new survey of approaches for detecting data corruption, or a detailed analysis of its impact on scientific computing applications — a variety of work in that space has in fact been done over the past several years, which we recommend for further reading [BGC15, DC16, Pei15a, Pei15b, Pei17, RVD+19, WA19]. For example, we do not focus on the questions of how many bits need to flip in order for a scientific result to be materially altered or a dataset rendered useless.

This document is primarily targeted at scientists and IT personnel involved in scientific computing applications.   It is secondarily targeted at personnel involved in maintaining "scientific cyberinfrastructure" such as community data repositories, computing facilities, and user facilities with scientific instruments, though we presume that such environments tend to have existing protocols, established over many years through scientific community dialogues, to determine best practices for things like data integrity.

2 Background

Binary notation is the base-2 representation of numbers as combinations of the digits 0 and 1, in contrast to the decimal notation most of us are used to in our daily lives that represents digits as combinations of the digits 0 through 9. In binary notation, A "bit" is the atomic element of the representation of a 1 or a 0. Bits — 0's or 1's — can be combined together to represent numbers larger than 0 or 1 in the same way that decimal digits can be put together to represent numbers larger than 9.

Binary notation has been in use for many hundreds of years. The manipulation of binary numbers made significant advances in the mid 19th century through the efforts of George Boole, who introduced what was later referred to as Boolean algebra or Boolean logic. This advance in mathematics, combined with electronic advances in switching circuits and logic gates by Claude Shannon (and others) in the 1930s led to binary storage and logic as the basis of computing. As such, binary notation, with numbers represented as bits, are the basis of how most computers have stored and processed information since the inception of electronic computers.

However, while we see the binary digits 0 and 1 as discrete, opposite, and rigid representations, in the same way that North and South represent directions, the components of a computer that underlie these 0 and 1 representations are analog components that reveal that 0 and 1 are in fact closer to shades of grey. In fact, 0 and 1 are typically stored magnetically and transmitted through electrical charges. In reality, both magnetism and electrical charges can either degrade or otherwise be altered through external forces, including cosmic rays or other forms of radiation and magnetism. To a computer, a "bit flip" is the change of the representation of a number from a 0 to a 1 or vice versa. Underlying that "flip" could have been a sudden burst of radiation that suddenly and instantly altered magnetic storage or electrical transmission, or could also have been the slow degradation of the magnetism of a magnetically stored bit from something close to 1, or a "full" magnetic charge, to something less than 0.5, at which point it would be recognized and interpreted as a 0.

The use of error correction in computing and communication was pioneered in the 1940s and 1950s by Richard Hamming, who used some form of redundancy to help to identify and mask the effects of bit flips. Despite the creation of these techniques 70–80 years ago, it is still not the case that error correction is universally used. And, even when it is, there are limits to the amount of errors that can be incurred in a particular blob of data (a number, a file, a database) before the errors can fail to be correctable, or even to be detected at all.

3 Current State of Practice

In some cases, such as an election, a single bit flip is clearly vital, since a single bit can determine if a race is won or lost by a given candidate. In other places, we have learned, in our survey of the community, that research computing teams tend to assume bit flips are dealt with at the hardware, network, and/or filesystem level, and simple checksums are generally trusted. Where those basic measures fail, bit flips are typically assumed to be possible, and, like most science data that contains "noise," are inconsequential, and simply lost in that noise. A good example is image data, which can frequently contain distortions or other anomalies, regardless of flipped bits. Alternatively, for a set of time series events, there's an assumption that if you don't have every single point, or some individual point is off, as is frequently the case with sensor data, you just throw that point away.

It is scientific cyberinfrastructure that tends to set the "gold standard" for addressing integrity. Consider DataONE and its constituent repositories — hashes and checksums are used, data is immediately replicated (with the hash of the replicated object checked upon replication), and the data is write-once-never-delete. Getting data into DataONE leverages the metacat software containing built-in integrity checks. Additionally, to defend against malicious alteration, access controls are in place for reads and writes, such as managing storage space and protecting against abuse. Even there, the write-once-never-delete policy addresses data modification and loss. Of course DataONE is a repository with preservation as a goal, and so it's at one end of the spectrum of measures taken to defend against modification or loss.

On the same end of the spectrum are science projects that have particular security needs, such science involving detection of atmospheric conditions hazardous to human health. Here, too, the "right" thing is usually done with access controls, data transfers via SSL/SCP (which leverage the encryption protocol's hash function), sanity checks of metadata, split storage of hashes and data, pushing data to multiple locations, and regular backups.

What about scientific research involving computing other than examples such as these? What do they do? Most other projects seem rarely to do things that require extra effort from researchers, or require challenging integrations, like hashing files on disk. On the other hand, checksums are built into a lot of tools like SSH and Globus. REST APIs may be the "wild west" -- while some API authors will use SSL, others may not. And of course FTP simply relies on TCP checksums, which, as we discuss below, may be insufficient. In our investigations, we discovered that ZFS is much more commonly used than we would have expected, as a backend by databases that lack a multi-phase commit (e.g., Postgres, mySQL) to avoid corruption. RAID-5 or above is typically a given, to also help avoid corruption and data loss. Both ZFS, a software solution, and RAID-5 (or above), a mixed hardware/software solution, represent important and relatively straightforward safeguards to data integrity from bit flips and data loss, respectively.

The projects that follow the most rigorous practices with regard to data loss and modification are typically data repositories with the primary goal of protecting data. However, DataONE, one such repository, only covers the repository, not the path from the sensor or the simulation *to* the repository.[1]

4 Example Use Cases

We begin by discussing two example use cases.

Automated Chemical Analysis

An automated chemical analysis workflow is an example of integrity practices that could stand to be improved, should the need be of importance.

The chemical analysis workflow begins with data being read by the network-connected instrument. Each data file from the sensor is about 800 MB and takes about 15 minutes to generate. This data is then manipulated and analyzed in an automated fashion and ultimately is stored in a database for user retrieval. At each step, data is transferred over an IP network and is handled by different servers. Bit errors on the data and files used in each step will have a different impact on the final results.

If the analog datastream from the instrument has sudden changes in the recorded value, results can be assumed to be incorrect. Researchers expect Gaussian-shaped signals on a noise floor. In order to validate that this is true, in the workflow, the shape of this function is checked and examined for sudden spikes (e.g., caused by a bit error). If such deviations occur, that data point is discarded, although no alert is given nor are logs kept of this action.

[1] Though not a focus of this report, we note in passing that for most projects, other than those involving the most sensitive of information, the notion of someone modifying data maliciously is not seen as a major risk other than basics of authentication and access controls.

The data elements containing time and masses have to be maintained in sequential order. However, the workflow does not currently check to see if the time and masses recorded are in order. The analysis scripts assume times and masses are in order but do not check. In addition to bit flips in data elements, bit flips in metadata could have dramatic consequences ranging from the misinterpretation of results to mixing different samples. This could be partially checked if the data is in the expected ranges (e.g., adequately spaced from each other), which is not currently being performed.

After the raw, sampled data from the chemical analysis instrument is analyzed, the resulting output is transformed into an area or height value. These values, unlike the raw, sampled data, do not have the "redundancy" present that enables error checking (i.e., if it fits the Gaussian shape), and therefore would suffer greatly from any bit flips; only major outliers would be found in the finalizing QA/QC process from a human operator. Data size is also dramatically reduced, from 800 MB to 1 KB.

We now discuss the protocols used and their impact on data integrity:

Data is first captured on the analysis instrument's built-in computer's hard drive. This computer with ECC memory runs a Windows 10 operating system. It is using the NTFS filesystem, which does not currently have built-in integrity properties like ZFS. From there, the raw data is converted from an internal file format to an industry-standard output format, though with the same values. This standard data format is stored on disk again and then transferred via Samba over an IP network to network storage, a Linux-based server also with ECC memory.[2] This network storage is then accessed by another machine via NFS, and with several processing operations, such as analyses, and conversions, in between, each with intermediate files stored on local disk. Finally the derived data is stored again on the same NFS system. From there, it is copied as a 'csv' file to the user's local system via NFS or Samba and uploaded from the user's local system to the database server.

This database server is only storing a few KB per experiment, resulting in the database being fairly small (~50MB). Apart from regular backups, the data can be restored from the original source data, which is currently backed up to a Google Drive via the Google Drive web interface (and so integrity is therefore theoretically maintained via HTTPS). This however would require doing significant and time consuming analysis work again.

[2] This server is currently being upgraded from RAID-6 to ZFS (array of mirrored vdevs) while receiving additional storage disks for increased capacity.

For both Samba and NFS storage, the protocol ensures that the file transfer is complete before reporting success. In addition, both methods could and should use encrypted file transfers that in addition to privacy also provide integrity protection based on the hash algorithm used by the encryption protocol. However, both use the default configuration of each protocol that uses unencrypted and non-integrity-protected file streams, thereby solely relying on the TCP/IP protocol checksums to check for bit errors.

If the project were to consider enhancing integrity protections, it should switch to encrypted file transfer as the effort of doing so would likely be complete in one or two hours. The project could also ensure greater integrity checking at relatively low cost by using ECC memory and ZFS at each point in the storage chain. Alternatively, cryptographic hashes could be used at each point in the process, instead of ZFS and/or encrypted communications. If bit flips were deemed to be frequent or particularly devastating, using a fault tolerant, consensus-based computing approach, such as Paxos, would also be warranted.

Urban Sensor Data Analysis

An urban sensor data analysis pipeline is a positive example of "integrity done right."

This project is a scientific data collection and retrieval system of data pertaining to detection of hazardous conditions with respect to air quality, radiation, etc.... It is used by dozens of sites (each with potentially many users at each site) and contains currently 280TB of data with datasets ranging from a few MB to hundreds of GB. Users have the ability to upload, search, and download these datasets using a simple web interface and also an API.

In the project, data integrity has been one of the design parameters from the beginning, and each step in the processing chain has built-in data integrity protection. As an example, the file transfers and process for user upload and download are described below.

The user can upload data is two ways: Either via the web interface or SCP. Both methods are encrypted and integrity is protected with the encryption protocol's hash function.

After the data is received, the associated metadata is checked to ensure it is present and follows the design schema. Subsequently, the data is hashed and the hash and the file itself are stored on separate systems such that complete file losses are detectable. This hash is available to the user and the user can check the uploaded and downloaded data against the hash to detect partial data losses, including bit flips.

Metadata is stored in a MongoDB database that is regularly backed up, protecting against total data loss but not protecting the integrity of the metadata. Furthermore the metadata is saved alongside with the stored data. Therefore, the database can be completely rebuilt if an error is found. However, finding such an error requires the user to check the data, either through the stored hashes or through other means.

5 Detailed Component Discussion

Processor/Internal Bus/External Cabling

The undetected bit error rate of both internal busses and external cabling is 10^{-12}, and is in line with the other devices and components connected to computers that are used in a computer. This rate is due to the added bits present in communication protocols that serve the purpose of error correction, among other things.

In practical terms, scientists improve the undetected bit error rate within processors, busses, and cables by cross-checking suitably large checksums. This checking can be done, for example, if one is looking for bit errors during computation after multiple runs on the same device, or on other devices if the validation takes place after sending or receiving data between systems.

Consensus algorithms, such as Paxos [Lam98] and Practical Byzantine Fault Tolerance [CL02] can mask errors in addition to detecting them, although at a cost of required redundancy,[3] slowdown, and the effort to modify code and/or workflows to leverage the consensus algorithms. Masking errors is most important where the consequence of small numbers of random errors is very high, or in situations where the random error rate is particularly high, such as in environments with increased amounts of ionizing radiation, e.g.., at high elevations or the Earth's polar regions (see "Data Integrity in Extreme Environments" section below).

RAM

There are numerous analyses that provide statistics about how often bit flips in DRAMs occur. However, none of them apply for all situations and applications. In one study, Schroder et al [SPW09] monitored the DRAM errors in the thousands of systems of the famous Google server-farm for a period of 2 1/2 years. Those servers were surely perfectly air-conditioned, dust-free and protected from radiations of all kinds. Still they came to the result of 25,000 to 70,000 FIT (failures per billion device hours) of "ECC correctable errors" per Megabit of DRAM. This failure rate converts into an average of one single-bit-error

[3] To mask f failures, required redundancy is 2f+1 for Paxos and most crash-tolerant protocols; and 3f+1 for most BFT protocols

every 14 to 40 hours per Gigabit of DRAM [IM]. The field study also explains that the error-rate increases by the age of the memory.

Brand new DRAM might not show any errors for weeks and months, but then the error-rate suddenly goes up. Uncorrectable errors could be double- or multi-bit errors or complete functional fails of the DRAM. These can all not be corrected, but are extremely rare.

Error-correcting code memory [ECC] can detect and correct errors of a single bit per 64-bits or detect up to 2 errors utilizing Hamming Codes. This requires 12.5% (1 bit of redundancy per byte of data) of additional memory cells to store said redundancy information. Since DRAM errors are highly susceptible to the environment and age of the chips, the chance of having an undetected error can be as large as 9% over a timespan of 3 years [Del97] and grows exponentially with age.

Reading files from memory and comparing a current hash with a stored hash can be done periodically to check for such errors, but requires significant overhead and can only be performed if the content of the file in RAM is not changing.

Finally, we note that though the current solution is likely overkill for bit flips, hardware trusted execution environments (TEEs), such as Intel SGX [Int], that provide memory encryption and integrity protection, may protect against bit flips as well

Long Term Storage

HDDs and SSDs

Many errors are detected and corrected by the hard disk drives and solid state devices using the ECC/CRC codes that are stored on disk for each sector. If the disk drive detects multiple read errors on a sector it may make a copy of the failing sector on another part of the disk by remapping the failed sector of the disk to a spare sector without the involvement of the operating system (though this may be delayed until the next write to the sector). This "silent correction" can be monitored using S.M.A.R.T. and tools available for most operating systems to automatically check the disk drive for impending failures by watching for deteriorating SMART parameters. This remapping is particularly important for SSDs as sectors age quickly with the number of bytes written, and often less than the raw storage is available to the user to mitigate this effect [Kingston].

In a recent study of 1.53 million disk drives over 41 months [BAA+07], Bairavasundaram et al. show that 400,000 blocks across 3,855 drives (0.25% of the drives) had checksum mismatches. This same statistic also applies to USB “thumb drives,” as they use the same

NAND-Chips as SSDs. One conclusion we might draw from this is that if bit flips occur on storage media, it is likely to be more than a single block that is flipped.

In order to gain more reliability one can use multiple drives in a RAID configuration where parts of the storage is used for redundancy. By default, RAID-1, RAID-5, and RAID-6 (and derivatives) protect against failed devices (but not bit errors) by allowing for a certain number of failed devices before data loss. RAID can be used to detect errors as well by manually triggering integrity checks or rebuilds. RAID-5 and RAID-6 can fix errors in addition to detecting them through the same process. The downside of manual detection is the potential to not capture errors until the validation period, and only discover until later that any derived data calculations made on the erroneous data in the interim are wrong.

An alternative to periodic hardware-based integrity checks provided by RAID 1, 5, and 6, is to leverage software checks provided by some file systems, such as Btrfs, HAMMER, ReFS, and ZFS [ZRA+10]. These can use internal data and metadata checksumming to detect silent data corruption. ZFS, for example, can use SHA-256 to validate data integrity, reducing the probability of undetectable and uncorrectable errors to less than 2^{-256} per file. This can also help identify failing storage devices or respective bus interconnects [Goe17]. In addition, if a corruption is detected and the file system uses integrated mechanisms that provide data redundancy, such as ZFS's "RAID-Z", such file systems can also reconstruct corrupted data in a transparent way [BtrFS]. This error correction takes place at the cost of requiring multiple disks, but provides strong guarantees against masking any random data integrity failures.

Tape

We note in passing that tape storage uses forward correcting codes to achieve error rates of 10e-19 for LTO7 storage systems; which is particularly important for long term (up to 30 years [Tap19]) archives, as those are typically not checked routinely [Sli15]. The correct storage environment, with a Temperature 61°F to 77°F (16°C to 25°C) with a Relative Humidity 20% to 50% is important to achieve the predicted lifetime [Tap19].

Here, too, redundancy can help not only detect errors but also to enable correction. To our knowledge, no automated techniques for correcting integrity errors from tape systems exist, making this process more labor intensive than keeping data in live storage on disks and relying on techniques such as RAID-Z, where possible.

Non-Atomic Operations and Available Disk Space

In any storage environment, data is written to the storage media block by block. This happens through a caching mechanism for performance reasons. In the event that the cache is not written to disk correctly (e.g, because the storage media is full or there is a power outage and the cache is lost [Note: Some but not all raid controllers and SSDs have built in battery buffers to write the cache to disk]) there will be incomplete data written to disk. This can lead to data corruption if data is only partially written or is even partially overwritten. The application writing the data to disk has to be able to deal with failures in a safe manner, for example writing new data blocks to disk first and then changing a pointer to the most recent block in an atomic operation. If anything goes wrong while writing the data, only the most recently written data will be affected, but not existing data.

PostgreSQL is ACID (Atomicity, Consistency, Isolation, Durability) compliant and theoretically ensures that all requirements are met. However, even with ACID databases, there can be implementation bugs. For example, consider a string of database writes that are individually atomic but experience a disk full situation partially through writing the entire string. In such a case, the database may be left in an uncertain state in which even audit logs cannot be analyzed to determine exactly what happened because the disk filled in the process. Some databases and indeed some filesystems take steps to avoid such errors by reserving a small portion of the disk to finish a limited set of operations even when the disk is otherwise presented as "full." Alternatively, some databases can project forward the result of a string of comments to determine if there is sufficient space to successfully execute all of them. However, there also exist many systems that do not take such steps and/or have bugs in the implementation. The safest conclusion is that data corruption remains a real and present threat in the event of a disk full error, and so a disk should likely never be filled past a certain threshold. Thresholds will vary depending on the size of the disk and the type of applications the system is being used for, but limiting used disk space to something on the order of 80-90% before adding additional disk capacity is not generally considered unreasonable.

Networking

In networking with TCP/IP there will be transmission errors. The TCP checksums can detect up to 15 flipped bits per packet. However, bit flips can exceed the detection accuracy of the TCP checksum. In fact, "Between 1 packet in 1,100 and 1 packet in 32,000 can fail the TCP checksum" [SP00]. "We conclude that the checksum will fail to detect errors for roughly 1 in 16 million to 10 billion packets" [SP00], or between 0.008% and 0.22% of incorrect transmission passed the TCP checksum in a real life test [PHS95].

Example

For FTP data transfers, including Globus's GridFTP, MTU is typically 1,500 bytes [ABK+05]. So, for a 100 TB data transfer, we might expect an average of 2,000,000-60,600,000 errors, from which in expectation 6-4,000 errors will not be detected by TCP checksums [SP00].

Mediation

One remediation is a 32 Byte CRC. The error detection rate is about 1 in 4 billion packets [PHS95], and is just slightly better than the TCP checksum. On UNIX, this remediation can be done by running the "cksum" program on either side of the network transfer. Another method is to utilize cryptographic checksums such as using SHA-1 before and after the transfer by using the "sha1sum" program. This additional error detection can cause I/O and computation overhead — as pointed out by Jung et al., "checksum overhead [for computation and I/O] can be anywhere between 30% and 100%" [JLKC19].

Alternatively, in the secure shell v2 (SSH), a 20 byte SHA-1 is used on every packet, resulting in an error detection rate of 1 in 3.8e25, which is approximately 13 orders of magnitude better than the TCP checksum, albeit at potentially much slower transfer rates achieved than by using GridFTP [ABK+05].

Data Integrity in Cloud Computing

A legitimate question about solutions for memory, storage, and data integrity might be, "What about using cloud providers, instead?"

Amazon S3 and Google Cloud Storage both guarantee 99.999999999% durability on object data storage [S3Data, GCPStorage], and for Amazon, during S3 PUT and PUT Object copy operations (but not during compute) [S3Data].   In order to transfer data into and out of S3 and to the EC2 compute the AWS tool 'cloudtrail' [EC2CT] can be used. It applies a SHA-256 on both sides of the transfer. In addition, a linked list like structure is used to track any changes to files and make sure entire changes are not lost.

Thus what remains, aside from long-term data storage, data transfer, and PUT and PUT Object copy operations, are integrity during compute, which includes the processor and memory. Computations on EC2 are done with ECC memory enabled [EC2FAQ], significantly increasing memory data integrity. The remaining issue is integrity during computation, an area that cloud computing doesn't especially make easier, because integrity during computation requires redundancy. At the same time both memory and cloud computation are no worse than non-cloud computing, and may even be improved given the standard use of ECC memory and the presumed optimality of the environmental controls in cloud data centers.

Data Integrity in Extreme Environments

Ionizing Radiation

(Ionizing) radiation, such as in space or near high energy equipment can have devastating effects on electrical systems. Experiments looking at data storage [Ngu05] find that all HDD's under test fail if the sum of exposed radiation is over 18 krad(Si). Further, radiation can also cause random bit flips in computation as electrons can be ionized from the radiation and be interpreted as a signal when none is present. This effect of incorrect interpretation is becoming even greater the smaller chips become — this because fewer and fewer electrons are required for such mis-interpretation. For this reason, shielding or extensive redundancy, such as via the crash-tolerant Paxos methods and/or the Byzantine fault tolerant methods mentioned in the "Processor" section above, is essential for high-radiation environments.

Corrosion

Corroding cables, because of corrosive chemicals (including salt water, for example), will slowly increase the resistance of communication and power delivery cables. As the cables corrode, signal levels become lower and lower until signals can no longer be interpreted. To discover that the data cannot be interpreted, error detecting codes are used, which are present in all modern communication system (see sections above).

HDDs on Moving Vehicles

Magnetic drive heads are mounted just a few nanometers above the magnetic platter and must never touch the platter. If the drive head is touching the platter, the head is deformed which will cause it to fail. For this reason, acceleration and vibrations should be avoided that could push the drive head into the magnetic platter. Further, accelerations and vibrations that cause positioning errors (i.e., data is read/written from/to the wrong location) should be avoided. Kelly found that position errors from vibrations occur above 0.22 Grms (root mean square acceleration) and permanent damage occurs over 0.68 Grms [Kel19]. Further, accelerations larger than 60G for an operating drive (350G non-operating) regardless of the time of impact causes permanent damage (e.g., because the drive is dropped) [Tom19].

Sound-Induced Errors

Disk errors due to sound have also been observed [John18]. In experiments conducted by Johnson Controls, performance reductions of up to 50% were observed due to errors that were correctable. Undetected and uncorrectable errors were not part of the experimental results reported, but it seems safe to assume that they could exist.

6 Summary

We conclude by summarizing the current situation with regard to bit flips due to natural faults, and highlighting several best practices to mitigate those.

Data integrity can be affected at every level of computer systems and communication between those systems.

The central point that we wish to communicate is that most of the time, data integrity can be relatively straightforward to address at low cost and effort. For data integrity during communications, leveraging tools that have cryptographic checksums built in, such as SSH and Globus largely solves the issue for communication. For data at rest, distributing the data in multiple independent locations, and leveraging a filesystem, such as ZFS, that checks for and mitigates integrity failures, alongside a hardware solution, such as RAID-5 (or above) that guards against disk failure, also largely solves the issue for storage. Alternatively, the use of cloud computing resources appears to provide similar protection. And of course data should be backed up alongside cryptographic hashes and periodically spot-checked to ensure no variation from the expected hash.

Now that the use of tools such as SSH, ZFS, RAID-5, and cloud computing resources such as AWS and GCP are common in many environments, and Globus is well entrenched in the scientific community, taking these steps has gone from being a time-consuming process to one that is relatively turnkey. The smallest scientific computing efforts that may be scraping by on computing equipment that is many years old may not be easily capable of leveraging these techniques. However, any project that cares about the data it collects and generates, and who can afford a little more time spent properly configuring the system and resources to ensure adequate data redundancy, should be implementing these steps, and will largely address data integrity due to natural faults by doing so. While rare exceptions will exist (e.g., environments in particularly high radiation or other highly rugged conditions) that require substantially more effort, those exceptions are certainly fewer in number, and can and should likely be addressed on an individual basis by reviewing specialized equipment made for specialized circumstances.

References

[ABK+05] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The Globus Striped GridFTP Framework and Server, SC'05, ACM Press, 2005. https://dl.acm.org/citation.cfm?id=1105819

[BAA+07] Bairavasundaram, L. N., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H., Goodson, G. R., & Schroeder, B. (2008). An analysis of data corruption in the storage stack. ACM Transactions on Storage (TOS), 4(3), 8. http://static.usenix.org/events/fast08/tech/full_papers/bairavasundaram/bairavasundaram.pdf

[BGC15] Leonardo Bautista-Gomez and Franck Cappello. Detecting silent data corruption for extreme-scale MPI applications. In Proceedings of the 22nd European MPI Users' Group Meeting, ACM, 2015. https://dl.acm.org/ft_gateway.cfm?id=2802665&type=pdf

[BtrFS] "How I Use the Advanced Capabilities of Btrfs", 2012. https://www.oracle.com/technical-resources/articles/it-infrastructure/admin-advanced-btrfs.html

[CL02]: Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. ACM Trans. Comput. Syst. 20, 4 (November 2002), 398-461. https://dl.acm.org/citation.cfm?id=571640

[DC16] Sheng Di and Franck Cappello. Adaptive impact-driven detection of silent data corruption for HPC applications. IEEE Transactions on Parallel and Distributed Systems, 27(10):2809–2823, 2016. https://ieeexplore.ieee.org/abstract/document/7393580/

[Del97] T.J. Dell. "A white paper on the benefits of chipkill-correct ECC for PC server main memory." IBM Microelectronics division, Vol. 11, pp. 1-23, 1997. https://pdfs.semanticscholar.org/747a/d718761b7d848a12e4f3a82aa0f46117a815.pdf

[ECC] What is ECC memory? https://www.crucial.com/usa/en/memory-server-ecc

[EC2CT] "Validating CloudTrail Log File Integrity", https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-log-file-validation-intro.html

[EC2FAQ] "Amazon EC2 FAQs", https://aws.amazon.com/ec2/faqs/

[Goe17] "Silent Data Corruption Is Real", 2017. https://changelog.complete.org/archives/9769-silent-data-corruption-is-real

[GCPStorage] Google Cloud Storage Documentation - Storage Classes. https://cloud.google.com/storage/docs/storage-classes

[IM] How often do ECC-correctable single-bit errors occur and how about double/multi-bit errors? https://www.intelligentmemory.com/support/faq/ecc-dram/how-often-do-ecc-correctable-single-bit-errors-occur-and-how-about-double-multi-bit-errors.php

[Int] Intel, "Intel® Software Guard Extensions." https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html

[John18] Johnson Controls, "Impact of Sound on Computer Hard Disk Drives and Risk Mitigation Measures," 2018.

[JLKC19] Eun-Sung JUNG, Si LIU, Rajkumar KETTIMUTHU, and Sungwook CHUNG. High-performance end-to-end integrity verification on big data transfer. IEICE TRANSACTIONS on Information and Systems, 102(8):1478–1488, 2019. https://www.jstage.jst.go.jp/article/transinf/E102.D/8/E102.D_2018EDP7297/_pdf

[Kel19] Brett Kelly https://www.ept.ca/features/everything-need-know-hard-drive-vibration/

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014. https://dl.acm.org/doi/abs/10.1145/2678373.2665726

[Kingston] "Understanding Over-provisioning (OP)." https://www.kingston.com/us/ssd/overprovisioning

[Lam98]: Leslie Lamport. 1998. The Part-Time Parliament. ACM Trans. Comput. Syst. 16, 2 (May 1998), 133-169. https://dl.acm.org/citation.cfm?doid=279227.279229

[Ngu05] D. N. Nguyen, S. M. Guertin and J. D. Patterson, "The Effect of Total Ionizing Dose Degradation on Laptop Hard Disks," Proceedings of the 8th

European Conference on Radiation and Its Effects on Components and Systems, 2006.

[Pei15a] Sean Peisert et al. ASCR Cybersecurity for Scientific Computing Integrity. Technical Report LBNL-6953E, U.S. Department of Energy Office of Science report, February 2015. https://www.osti.gov/servlets/purl/1223021

[Pei15b] Sean Peisert et al. ASCR Cybersecurity for Scientific Computing Integrity — Research Pathways and Ideas Workshop. Technical Report LBNL-191105, U.S. Department of Energy Office of Science report, September 2015. https://www.osti.gov/servlets/purl/1236181

[Pei17] Sean Peisert. Security in High-Performance Computing Environments. Communications of the ACM (CACM), 60(9):72–80, September 2017. https://dl.acm.org/ft_gateway.cfm?id=3096742&type=pdf

[PHS95] Partridge, Craig, Jim Hughes, and Jonathan Stone. "Performance of checksums and CRCs over real data." In ACM SIGCOMM Computer Communication Review, vol. 25, no. 4, pp. 68-76. ACM, 1995. http://ccr.sigcomm.org/archive/1995/conf/partridge.pdf

[RVD+19] Mats Rynge, Karan Vahi, Ewa Deelman, Anirban Mandal, Ilya Baldin, Omkar Bhide, Randy Heiland, Von Welch, Raquel Hill, William L Poehlman, et al. Integrity Protection for Scientific Workflow Data: Motivation and Initial Experiences. In Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning). ACM, 2019. https://dl.acm.org/ft_gateway.cfm?id=3332222&type=pdf

[S3Data] “Data Protection in Amazon S3”, https://docs.aws.amazon.com/AmazonS3/latest/dev/DataDurability.html

[Sli15] “Tape update: LTO-7 bit error rate improves, LTFS use rises”, 2015. https://searchdatabackup.techtarget.com/news/4500254525/Tape-update-LTO-7-bit-error-rate-improves-LTFS-use-rises

[SP00] J. Stone and C. Partridge, “When The CRC and TCP Checksum Disagree”, Proc. ACM SIGCOMM, 2000. http://conferences.sigcomm.org/sigcomm/2000/conf/paper/sigcomm2000-9-1.pdf

[SPW09] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. "DRAM errors in the wild: a large-scale field study." ACM SIGMETRICS Performance

Evaluation Review. Vol. 37. No. 1. ACM, 2009.
https://dl.acm.org/citation.cfm?id=1555372

[Tap19] TapeOnline
https://tapeonline.com/lto-7-faq#whats-the-archival-life-of-lto-7-media

[Tom19] https://www.tomshardware.com/reviews/a-sturdy-companion,758-2.html

[WA19] Von Welch and Ishan Abhinit. Data integrity threat model. Technical report, 2019. http://hdl.handle.net/2022/23225

[ZRA+10] Zhang, Yupu, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. "End-to-end Data Integrity for File Systems: A ZFS Case Study." In FAST, pp. 29-42. 2010.
http://static.usenix.org/events/fast10/tech/full_papers/fast10proceedings.pdf#page=37

Version History

December 20, 2019, v0.1 — Preliminary Public Report
January 27, 2020, v0.11

November 11, 2020, v0.13 — Revised Final Report