Characterizing Impacts of Storage Faults on HPC Applications: A Methodology and Insights

Bo Fang,*‡ Daoce Wang,†‡ Sian Jin,† Quincey Koziol,§ Zhao Zhang,¶
Qiang Guan,‖ Suren Byna,§ Sriram Krishnamoorthy,*† Dingwen Tao†**
* Pacific Northwest National Laboratory, Richland, WA, USA
† Washington State University, Pullman, WA, USA
§ Lawrence Berkeley National Laboratory, Berkeley, CA, USA
¶ Texas Advanced Computing Center, Austin, TX, USA
‖ Kent State University, Kent, OH, USA

***Abstract*—In recent years, the increasing complexity in scientific simulations and emerging demands for training heavy artificial intelligence models require massive and fast data accesses, which urges high-performance computing (HPC) platforms to equip with more advanced storage infrastructures such as solid-state disks (SSDs). While SSDs offer high-performance I/O, the reliability challenges faced by the HPC applications under the SSD-related failures remains unclear, in particular for failures resulting in data corruptions. The goal of this paper is to understand the impact of SSD-related faults on the behaviors of complex HPC applications. To this end, we propose FFIS, a FUSE-based fault injection framework that systematically introduces storage faults into the application layer to model the errors originated from SSDs. FFIS is able to plant different I/O related faults into the data returned from underlying file systems, which enables the investigation on the error resilience characteristics of the scientific file format. We demonstrate the use of FFIS with three representative real HPC applications, showing how each application reacts to the data corruptions, and provide insights on the error resilience of the widely adopted HDF5 file format for the HPC applications.**

I. Introduction

In recent years, the increasing complexity in scientific simulation and emerging demands for training heavy artificial intelligence (AI) models require massive and fast data accesses, which urges high-performance computing platforms to equip more advanced storage infrastructures. To this end, flash-based solid-state drives (SSDs) have been widely employed in HPC systems as a replacement of hard disk drives (HDDs) to achieve an order of magnitude speedup in data access. Prior studies [1]–[3] showed that this rapid adoption of SSD-based I/O components, however, raises a new challenge to the overall HPC reliability. As shown in [1], uncorrectable bit error rate (UBER) of data center SSDs are between 10^{-11}, 10^{-9}, which results in a collective high error rate on the large-scale HPC system and breaks the JEDEC 2016 requirement for enterprise class (¡10^{-16}) [4]. Such concern is expected to be continuously present due to the fact that the sources of SSD failures, i.e., cell wear-out, program/read disturb errors, retention errors, power faults, radiations, etc, would not dissolve shortly.

There are two classes of failures prominently experienced on SSDs: fail-stop and partial disk failures. Unlike the fail-stop failures that cause the SSDs to become inaccessible from the higher level of the stack, partial drive failures only affect a portion of the SSD operations and the device remains to work from the user's perspective [5]–[12]. The consequence of the partial failures is severe: they may cause corrupted data on SSDs and trigger issues in the file system and application layers in the I/O stack. Therefore, HPC applications need to mitigate the impact of the partial disk failures and tolerate the data corruption propagation due to such failures from SSDs.

Mitigating data corruption in an HPC application has been a challenging task that requires efficient and effective solutions. Towards this goal, a wide spectrum of studies [13], [14] employ statistical fault injections to characterize the impact of hardware faults, and take the characteristics obtained on per application basis to guide the design of the fault tolerance strategy. However, since the common fault models investigated in those studies are bit-flip faults affecting computational units and memory, their insights are not indicative to reason about the application's error resilience characteristics against SSD-related data corruption, which demands a new systematic characterization approach.

The goal of this paper is to provide methodology for characterizing how different types of SSD-related faults would affect the behaviors of the HPC applications. We mimic the impact of partial disk failures on applications by introducing faults via an application's I/O operations and observe the outcomes of the applications after the fault injection. We introduce a fault injection framework, FFIS[1] (FUSE-based Fault Injection for Storage) that leverages the FUSE [15] interface to systematically inject faults into the applications' I/O path. FFIS supports multiple fault models, each of which represents a specific data corruption scenario observed from the partial disk failures. FFIS offers a uniform interface for users to apply fault injection campaigns on various real HPC applications, without any modification on the applications.

FFIS is built based on the following key assumptions: *(i)* we focus on the data corruption that manifests on the

‡These authors contributed equally.
**Corresponding author: Dingwen Tao (dingwen.tao@wsu.edu).

[1]https://github.com/FabioGrosso/pFsysFI.

application level, such as bit flips, shorn writes, etc. We explain the details of these faults in Section IV.*(ii)* those errors can further propagate beyond the file system layer [16]–[18], skip the verification mechanisms (e.g., fsck [19]) and silently compromise the data integrity of the applications.

Enabled by FFIS, systematic fault injection studies can be performed to reveal error resilience characteristics of an application or the common components exercised across applications: *(i)* FFIS is able to evaluate different inherent error masking capabilities for different applications, or to measure such ability of each phase of an application. This suggests a potential trade-off space for HPC systems to lower the requirement of the SSDs' reliability for faster data access while maintaining the same level of the overall application's reliability; *(ii)* as the modern HPC applications tend to leverage the scientific file format for efficient data management, FFIS is able to investigate how the certain scientific file format library handles the storage errors affecting both the file metadata and application data, thus to obtain the possible common error resilience characteristics of the applications while operating on such file format. This paper makes the following contributions:

- We design and build a fault injection framework, called FFIS, to model SSD-related failures at software level and to inject such faults systematically into HPC applications.
- We apply FFIS on three real-world HPC applications through comprehensive, large-scale fault injection experiments. Our evaluation shows that applications exhibit distinct error resilience characteristics for different fault models, and we offer the detailed explanation for each application's unique resilience characteristics.
- We unveil application-specific behaviors operating on the most widely adopted scientific file format - HDF5, and show the fault-tolerance behaviors of the HDF5 library against errors affecting the HDF5 metadata. We identify certain fields in the metadata that may cause SDC outcomes when affected by faults, and provide the solutions for auto-correction. To the best of our knowledge, it is the first research effort to systematically characterize the detailed error resilience of the HDF5 file format.

II. Background

In this section, we describe the key context for our study and its importance to motivate the FFIS framework development.

Fault, error, and failure: As described in [20], a (hardware) fault, error, and failure chain is defined as the following events: "a service is a sequence of a system's external states, a service **failure** means that at least one (or more) external state of the system deviates from the correct service state. The deviation is called an **error**. The adjudged cause of an error is called a **fault**". Below we specify these terms in the different context:

- *Storage system*: an SSD's partial failures refer to the events where SSDs are not providing the expected behavior or outcomes such that the SSDs' internal states are left with flipped bits or shorn data, and what causes such failures are considered as hardware faults, including power faults or ratification faults, etc..
- *File system*: the file system failures refer to the unsuccessful file operations such as I/O errors returned to the application, which can result from the storage failures, or software bugs (not covered in this study).
- *Application*: a failure of an application refers to that scenario that the outcome of the application differs from the expected: the application either terminates before it finishes (i.e., *crash*), or it suffers from data corruption. If the application is able to identify the errors, this failure is categorized as *detected*, otherwise such data corruption becomes silent data corruption (SDC).

FUSE File system in user-space is the most widely used user-space file system framework [15] on Unix/Linux systems. It exposes the file operations to the file system users and allows the users to implement their own file operations such as *open(), read(), write(),etc.* if needed. When a FUSE file system is in use, the programs are able to access the data using the standard file operation system calls (i.e., POSIX) through the implementation of those system calls in the FUSE.

Why choose FUSE We choose FUSE as the underline file system interface for the fault injection framework for two reasons: *(i)*, as the goal is to study the impact of the failures on applications, the fault injection framework focuses on mimicking the SSD-related faults occurring during the I/O operations in the application level. FUSE allows us to implement such faults with a relatively straightforward manner;*(ii)*, since a FUSE-based file system works as the callbacks for the file operations, it allows the applications to call the user-implemented I/O primitives without modification on either the application or the actual running file system. This releases the burden for HPC applications that usually consists of complex behaviors and conservative execution environment.

Manifestation of SSD failures Zheng et al. [21] found that the power faults can cause a large number of SSDs to fail partially and produce the number of chip-level bit errors that exceed the correction capability of SSDs' error correcting code (ECC) and bypass the SMART (Self-Monitoring, Analysis and Reporting Technology) system [22]. For example, A recent study [1] reports that on SSDs the occurrence of partial drive failures that lead to data corruption can be an order of magnitude higher than on HDDs [1]. As shown in [16], this type of the SSD failures can manifest in the file system and affect the application's I/O behaviors. This leads to two classes of failures in general: *(a)* the file system throws the I/O errors and leaves the handling to the application and the typical failures include uncorrectable bit corruption, metadata corruption, incomplete I/O operations; *(b)* the file system does not detect such failures and the failures may cause data corruption in the application. These failures include silent bit corruption (bit flips in the data), shorn write (a write operation is partially done on the device), and dropped write (the file system issues the write but never gets executed on the device), which is the focus of this paper.

Why focusing on HDF5 file format Hierarchical Data Format version 5 (HDF5) provides an API for performing I/O, data management tools, and a portable file format [23]. In